PATENT ABSTRACTS OF JAPAN

(11)Publication number : 11-328134
(43)Date of publication of application : 30.11.1999

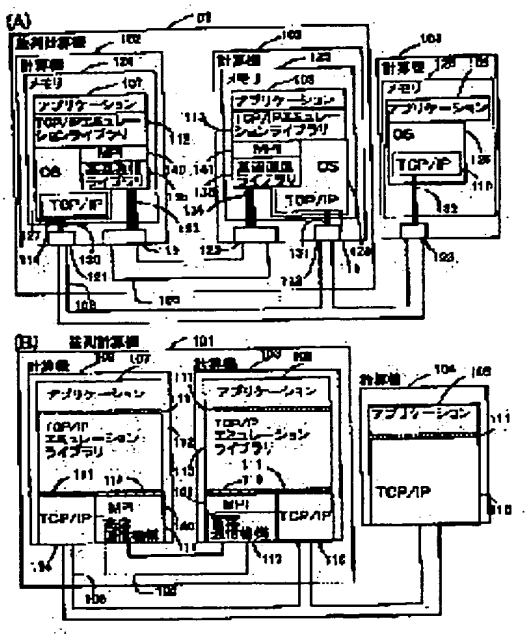
(51)Int.Cl. G06F 15/163
G06F 13/00

(21)Application number : 10-132364
(22)Date of filing : 14.05.1998

(71)Applicant : HITACHI LTD
(72)Inventor : IMAKI TSUNEYUKI
SAGAWA NOBUTOSHI

(54) METHOD FOR TRANSMITTING AND RECEIVING DATA BETWEEN COMPUTERS

(57)Abstract:
PROBLEM TO BE SOLVED: To utilize a message passing type stream communication from a TCP/IP application program.
SOLUTION: An emulation library 112 provided for each computer decides whether or not a communication party process that a communication request from an application program 107 specifies is inside or outside a parallel computer 101 and when the process is inside the computer, a fast communication library 135 is actuated to transmit and receive data by a message passing type communication system through a fast internal communication network 105. The data are transmitted and received so as to actualize a stream communication. When the communication party is an external computer 104 of the parallel computer 101, the library 112 requests a communication of a TCP/IP process routine 114 according to TCP/IP.



LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

(19)日本国特許庁（ＪＰ）　(12)**公開特許公報**（Ａ）　(11)特許出願公開番号

特開平11－328134

(43)公開日　平成11年(1999)11月30日

(51)Int.Cl.[6]		識別記号	FI		
Ｇ０６Ｆ	15/163		Ｇ０６Ｆ	15/16	３１０Ｒ
	13/00	３５１		13/00	３５１Ｅ

審査請求　未請求　請求項の数12　ＯＬ　（全 23 頁）

(21)出願番号　特願平10－132364

(22)出願日　平成10年(1998)５月14日

(71)出願人　000005108
株式会社日立製作所
東京都千代田区神田駿河台四丁目６番地

(72)発明者　今木　常之
東京都国分寺市東恋ケ窪一丁目280番地
株式会社日立製作所中央研究所内

(72)発明者　佐川　暢俊
東京都国分寺市東恋ケ窪一丁目280番地
株式会社日立製作所中央研究所内

(74)代理人　弁理士　高橋　明夫　（外１名）

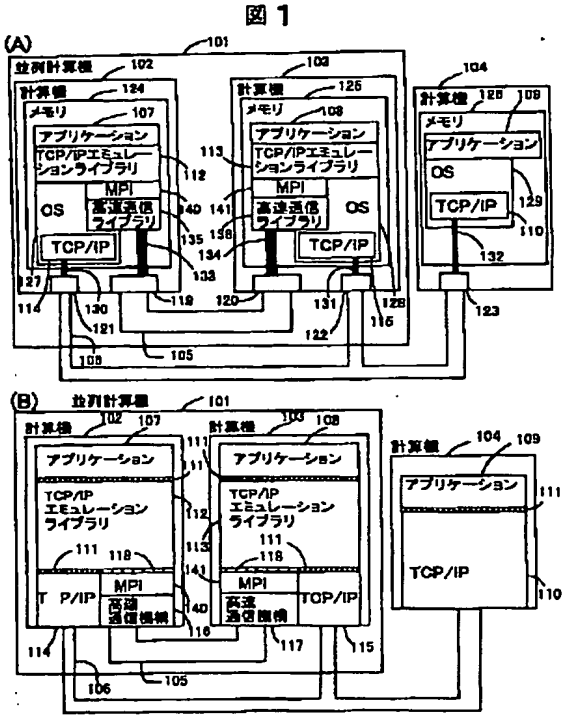
(54)【発明の名称】　計算機間データ送受信方法

(57)【要約】

【課題】ＴＣＰ／ＩＰアプリケーションプログラムからメッセージパッシング型のストリーム通信を利用可能にする。

【解決手段】各計算機に設けられたエミュレーションライブラリ１１２が、アプリケーションプログラム１０７からの通信要求が指定する通信相手プロセスが並列計算機１０１の内部であるか外部であるかを判別し、内部であれは高速通信ライブラリ１３５を起動して、高速内部通信網１０５を介してメッセージパッシング型通信方式でもってデータを送受信する。このデータの送受信は、ストリーム通信を実現するように実行される。相手先が並列計算機１０１の外部計算機１０４であるときには、ライブラリ１１２は、ＴＣＰ／ＩＰ処理ルーティン１１４にＴＣＰ／ＩＰに従い通信を要求する。

図１



【特許請求の範囲】

【請求項１】第１の計算機上で実行されている第１のアプリケーションプログラムが発行した複数の送信命令により指定される複数の送信データのそれぞれを、第２の計算機上で実行されている第２のアプリケーションプログラムが発行した複数の受信命令に応答して、第２の計算機上に設けられたエミュレーションライブラリの制御下でメッセージパッシング型の通信で受信し、

上記複数の送信データの連なりからなるひと繋がりのデータの内、上記複数の受信命令により指定されるサイズ部分に区分して得られる部分をそれぞれの受信命令が指定する複数のバッファに格納するように、受信されたそれぞれの送信データを上記エミュレーションライブラリの制御下で処理する計算機間データ送受信方法。

【請求項２】メッセージパッシング型の通信を実行するための第１の通信ライブラリを有する第１の計算機上で実行されている第１のアプリケーションプログラムが発行した複数の送信命令に応答して、それらの送信命令により指定される複数のバッファに保持された複数の送信データの送信をそれぞれ要求する複数の送信命令を、上記第１の計算機に設けられた第１のエミュレーションライブラリにより上記第１の通信ライブラリに対して発行し、

メッセージパッシング型の通信を実行するための第２の通信ライブラリを有する第２の計算機上で実行されている第２のアプリケーションプログラムが発行した複数の受信命令に応答して、上記複数の送信データをメッセージパッシング型の通信で受信するための複数の受信命令を、上記第２の計算機に設けられた第２のエミュレーションライブラリにより上記第２の通信ライブラリに対して発行し、

第１のエミュレーションライブラリが発行した上記複数の送信命令により指定される複数の送信データの連なりからなるひと繋がりのデータを、上記複数の受信命令により指定されるサイズ部分に区分してそれぞれの受信命令が指定する複数のバッファに格納するように、第２のエミュレーションライブラリが発行した上記複数の受信命令により受信される上記複数の送信データの第２の計算機内での格納位置およびそれぞれの送信データの受信後の第２の計算機内での移動を、上記第２のエミュレーションライブラリにより制御する計算機間データ送受信方法。

【請求項３】メッセージパッシング型の通信を実行するための第１の通信ライブラリを有する第１の計算機上で実行されている第１のアプリケーションプログラムが発行した複数回の送信命令に応答して、それらの送信命令により指定される複数のバッファに保持された複数の送信データの送信をそれぞれ要求する複数の送信命令を、上記第１の計算機に設けられた第１のエミュレーションライブラリにより上記第１の通信ライブラリに対して発行し、

メッセージパッシング型の通信を実行するための第２の通信ライブラリを有する第２の計算機上で実行されている第２のアプリケーションプログラムが発行した複数の受信命令に応答して、上記複数の送信データをメッセージパッシング型の通信で受信するための複数の受信命令を、上記第２の計算機に設けられた第２のエミュレーションライブラリにより上記第２の通信ライブラリに対して発行し、

第１のエミュレーションライブラリが発行した上記複数の送信命令が指定する複数の送信データを連ねて構成されるひと繋がりのデータを、受信側のアプリケーションプログラムが発行した複数回の受信命令により指定される複数のバッファに、それぞれの受信命令が指定するサイズ部分に区分して格納できるように、上記第２のエミュレーションライブラリにより（ａ）第２のエミュレーションライブラリが発行した上記複数の受信命令の各々が指定する、受信データを格納するバッファのアドレスを、現在処理中の第２のアプリケーションプログラムが発行した受信命令が指定するバッファに属するアドレスもしくは上記第２の通信ライブラリが利用可能なバッファに属するアドレスとなるように定め、かつ（ｂ）、上記第２の通信ライブラリが利用可能な上記バッファのアドレスを指定する、第２のエミュレーションライブラリが発行した受信命令により受信した送信データのその後の上記第１の計算機内での移動を制御し、

第２のエミュレーションライブラリが発行した上記複数の受信命令は、それぞれ第１のエミュレーションライブラリが発行した上記複数の送信命令の一つに対応して発行され、

第２のエミュレーションライブラリが発行した上記複数の受信命令の各々が指定する受信データのサイズは、第１のエミュレーションライブラリが発行した上記複数の送信命令の内の対応する一つの送信命令が指定する送信データのサイズに等しくなるように設定される計算機間データ送受信方法。

【請求項４】（ａ）送信側のアプリケーションが指定するいずれかの送信データの送信を要求する、上記メッセージパッシング型の通信により定められた送信命令を送信側の計算機内で発行し、（ｂ）上記送信命令が指定する送信データ長を、その送信データの送信前に、受信側の計算機上で動作する受信側のエミュレーションライブラリにより検知し、（ｃ）受信側のアプリケーションが発行した受信命令によって指定された受信データ長と上記送信データ長とを、上記受信側のエミュレーションライブラリにより比較し、（ｄ）上記比較の結果に応じて、上記受信側の計算機内のエミュレーション用のバッファに上記送信データを受信することを要求するかもしくは上記送信データを上記アプリケーションバッファに受信することを要求する、上記メッセージパッシング型

の通信により定められた受信命令を上記受信側のエミュレーションライブラリにより発行し、（ｅ）上記送信命令と受信側のエミュレーションライブラリにより発行した上記受信命令とに応答して、上記送信データを上記送信側の計算機と上記受信側の計算機により、上記送信側の計算機から上記エミュレーション用のバッファもしくは上記アプリケーションバッファ内に転送する計算機間データ送受信方法。

【請求項５】（ｆ）上記送信データが上記エミュレーションバッファに送信されたときには、そのエミュレーションバッファから上記受信データ長分だけのデータを上記受信命令が指定したアプリケーションバッファ内に、上記受信側のエミュレーションライブラリによりコピーするステップをさらに有する請求項４に記載の計算機間データ送受信方法。

【請求項６】（ｇ）上記ステップ（ｅ）により、上記送信データが上記アプリケーションバッファ内に転送された場合、上記送信側の計算機が後続の送信データに関してステップ（ａ）を実行するのを待ち、（ｈ）上記後続の送信データに関して、ステップ（ａ）が実行された場合、その後続の送信データと上記受信命令が要求した受信データ長の残りの長さのデータに関して上記ステップ（ｂ）以降のステップを実行するステップをさらに有する請求項５に記載の計算機間データ送受信方法。

【請求項７】（ｉ）上記ステップ（ａ）を実行する前に、上記エミュレーションバッファ内に受信済みでいずれのアプリケーションバッファにも転送されていないデータが残っているか否かを判別し、（ｊ）そのデータが残っているときには、そのバッファから上記アプリケーションに上記受信データ長を超えない範囲でコピーするステップをさらに有する請求項６に記載の計算機間データ送受信方法。

【請求項８】第１の通信網で相互に結合されている複数の計算機を有し、少なくともその一部の計算機が第２の通信網に相互に接続されている計算機システムにおいて、（ａ）上記一部の計算機の内の第１の計算機上で動作する第１のアプリケーションプログラムから、第２の計算機上で動作するいずれかの第２のアプリケーションプログラムとの間のデータ送受信に使用する通信路を、第１の通信網を使用するために定められた通信規約に従って動作する処理ルーチンにより決定し、（ｂ）第１のアプリケーションプログラムからの、上記決定された通信路に対する接続要求をエミュレーションライブラリにより処理し、その処理の中で、上記第２のアプリケーションプログラムとの通信に上記第２の通信網を利用可能か否かを判別し、（ｃ）上記第２の通信網が利用可能と判別されたときには、上記通信路を上記エミュレーションライブラリ内に登録し、（ｄ）その後上記第１のアプリケーションプログラムがデータ送受信命令を発行したときに、その送受信命令により指定された通信路が、上記エミュレーションライブラリ内に登録されているか否かを判別し、（ｅ）その指定された通信路が、上記エミュレーションライブラリ内に登録されているときには、上記エミュレーションライブラリにより、上記送受信命令が要求する送受信処理を上記第２の通信網を使用して実行し、（ｆ）その指定された通信路が、上記エミュレーションライブラリ内に登録されていないときには、上記エミュレーションライブラリにより、上記送受信命令が要求する送受信処理の実行を上記処理ルーティンに要求し、上記処理ルーチンにより、その送受信処理を上記第１の通信網を使用して実行するデータ送受信方法。

【請求項９】上記通信規約はＴＣＰ／ＩＰであり、
上記通信路の決定は、ソケットの生成およびソケットへの名前付けの処理を含む請求項８記載のデータ送受信方法。

【請求項１０】上記第２の通信網を使用した通信はメッセージパッシング型の通信である請求項８記載のデータ送受信方法。

【請求項１１】第１の通信網で相互に結合されている複数の計算機を有し、少なくともその一部の計算機が第１の通信網よりも高速にデータを転送可能な第２の通信網に相互に接続され、上記一部の計算機の内の第１の計算機上で動作する第１のアプリケーションプログラムから、第２の計算機上で動作するいずれかの第２のアプリケーションプログラムとの間のデータ送受信に、上記第２の通信網を使用可能であるときには、これを使用し、そうでないときには上記第２の通信網を使用する計算機システムにおいて、
第１のアプリケーションプログラムに宛てて他のアプリケーションプログラムから送信されるデータの有無を検出するときに、上記第２の通信網を介して送信されるデータの検出処理と、上記第１の通信網を介して送信されるデータの検出処理とを、ノンブロックで交互に繰り返し、
両検出処理を１回繰り返す毎に両検出処理を中断し、上記第１の計算機上で動作している他のプロセスに処理を譲渡する送信データ検出方法。

【請求項１２】第１の通信網で相互に結合されている複数の計算機を有し、少なくともその一部の計算機が第１の通信網よりも高速にデータを転送可能な第２の通信網に相互に接続され、上記一部の計算機の内の第１の計算機上で動作する第１のアプリケーションプログラムから、第２の計算機上で動作するいずれかの第２のアプリケーションプログラムとの間のデータ送受信に、上記第２の通信網を使用可能であるときには、これを使用し、そうでないときには上記第２の通信網を使用する計算機システムにおいて、
第１のアプリケーションプログラムに宛てて他のアプリケーションプログラムから送信されるデータの有無を検出するときに、二つのスレッドを生成し、

上記第２の通信網を介して送信されるデータの検出処理と、上記第１の通信網を介して送信されるデータの検出処理とを、それぞれのスレッド上で独立に行なうデータ検出方法。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は、複数の種類の通信網により接続された複数の計算機を有する計算機システムにおける、計算機間のデータ送受信方法に係り、特にメッセージパッシング型の通信を実行するのに好適な、計算機間のデータ送受信方法に関する。

【０００２】

【従来の技術】計算機間の通信規約としてＴＣＰ／ＩＰが極めて一般的に使用されている。ＴＣＰ／ＩＰを使用して他のプログラムと通信するように構成されたプログラムを以下ではＴＣＰ／ＩＰアプリケーションプログラムと呼ぶ。このＴＣＰ／ＩＰアプリケーションプログラムからＴＣＰ／ＩＰ以外の通信規約を使用できるようにした計算機システムも存在する。すなわち、計算機システムを構成する計算機が性能の異なる複数のネットワークにより構成され、そのシステムの内の計算機が他の計算機と通信するときに、それらの計算機がその通信にこれらのネットワークのいずれを使用するかに依存して、ＴＣＰ／ＩＰまたは他の通信規約を使用するシステムも提案されている。例えば、バークレー大のSteven H. Rodriguesらが開発したシステムは、ＴＣＰ／ＩＰプロトコルで使用可能な広域のネットワークと、それより狭い領域でより簡潔でオーバヘッドが低い通信規約を使用する局所的なネットワークからなり、局所的なネットワークに接続された計算機同士は、この通信規約を使用して通信し、広域のネットワークに接続された計算機同士は、ＴＣＰ／ＩＰ規約に従って通信する。たとえば、"High-Performance Local-Area Communication With Fast Socket", USENIX '97 Annual Technical Conference pp. 257-274)参照。

【０００３】この計算機システムには、ＴＣＰ／ＩＰアプリケーションプログラムからもこの通信規約を利用できるようにするために、ＴＣＰ／ＩＰエミュレーションライブラリが用意されている。上記システムではこのＴＣＰ／ＩＰエミュレーションライブラリは、高速ソケット（Ｆａｓｔ　Ｓｏｃｋｅｔ）と呼ばれている。

【０００４】高速ソケットは、簡潔な通信規約としてワークステーションクラスタ向けに開発されたアクティブメッセージ（Ａｃｔｉｖｅ　Ｍｅｓｓａｇｅ）を用いる。例えば 文献 T. von Eicken, D. E. Culler, S. C. Goldstein, K. E. Schauser "Active Messages: a Mechanism for Integrated Communication and Computation", in Proceedings of the 19th International Symposium on Computer Architecture, May 1992 pp. 256-266 参照。アクティブメッセージでは、データを送る側のアプリケーションプログラムが受け手側のアプリケーションプログラムに割り込み、受け手側がその割り込みを契機にデータ受信処理を行うという方法でデータをやり取りする。

【０００５】並列計算機は、複数の計算機が互いに通信しながら協調して１つの問題を解決することを目的とした計算機である。この目的を満たすため、一般に並列計算機の内部の各計算機は高速の内部高速通信網により相互に接続され、並列計算機内の少なくとも一部の計算機は、さらに、ＬＡＮ等のより広域のネットワークにより外部の計算機に接続されている。

【０００６】広域のネットワークに使用される通信規約は主としてＴＣＰ／ＩＰである。並列計算機は、内部高速通信網を使用するために、各計算機に他の内部計算機と内部高速通信網を介して通信するための高速な通信ハードウェアとそのハードウエアを使用するための高速通信ライブラリを設けている。現在、多くの並列計算機に採用されている通信規約は、メッセージパッシング型の通信である。メッセージパッシング型の通信は、送信アプリケーションプログラムが発行する送信命令と受信アプリケーションプログラムが発行する受信命令が一対一に対応付けられた時に通信が行われるという通信である。多くの場合、この通信方式は並列計算機内部における高速通信ハードウェアに適している。このメッセージパッシング型通信を実現するために使用される高速通信ライブラリには、リモートメモリ書き込みライブラリあるいはＰＵＴライブラリと呼ばれているものが主に使用されている。

【０００７】並列計算機内部の通信のオーバヘッドは、高速通信ハードウェアを使用することによりかなり減じることができる。アクティブメッセージで用いる割り込み型の通信方式では、割り込みのオーバヘッドが目立ってしまう。しかるに、メッセージパッシング型通信方式は、割り込みを前提としないので、並列計算機の内部での通信にはアクティブメッセージ型通信よりも適している。

【０００８】

【発明が解決しようとする課題】前述のように、並列計算機の内部高速通信網を使用するための通信規約は一般にはメッセージパッシング型である。しかし、並列計算機をビジネスの分野で利用する場合に使用するビジネス用のアプリケーションプログラムは多くはＴＣＰ／ＩＰ規約を使用するように構成されている。したがって、そのようなアプリケーションプログラムは、そのままではメッセージパッシング型通信方式をそのまま利用することはできない。しかも、メッセージパッシング型通信方式を利用した状態で、ストリーム通信を実現する方法が知られていない。また、割込を使用するアクティブメッセージ型通信で使用されるストリーム通信の実現方法をそのままメッセージパッシング型通信でのストリーム通

信に利用するわけには行かない。

【０００９】従って本発明の目的は、メッセージパッシング型通信を実行するように構成されている計算機上で動作する複数のアプリケーションプログラム間でストリーム通信を実行可能にする計算機間データ送受信方法を提供することである。

【００１０】本発明のより具体的な目的は、第１の計算機ネットワークを介して、ＴＣＰ／ＩＰのごとき、メッセージパッシング型通信と異なる通信規約を用いて通信を実行するように構成されている複数のアプリケーションプログラム間で、その計算機ネットワークよりも高速な通信網を使用して、かつ、メッセージパッシング型通信を実行可能にする計算機間データ送受信方法を提供することである。

【００１１】さらに、本発明の他の目的は、第１の通信網とそれより高速の第２の通信網に接続された計算機上で動作し、他のアプリケーションプログラムとの間で、たとえばＴＣＰ／ＩＰ通信規約のような第１の通信規約に基づいて通信するように構成されたアプリケーションプログラムが、第１の通信網に接続された他の計算機上で動作する他のアプリケーションプログラムとの間で、その通信規約に基づいた通信を実行することを可能にするとともに、第２の通信網に接続された他の計算機上で動作する他のアプリケーションプログラムとの間でその第２の通信網を使用した高速の通信を実行する行うことを可能にする計算機間データ送受信方法を提供することである。

【００１２】さらに、本発明の他の具体的な目的は、上記第２の通信網を使用した通信を、メッセージパッシング型の通信とすることができる計算機間データ送受信方法を提供することである。

【００１３】

【課題を解決するための手段】上記目的を達成するために、本発明による計算機間データ送受信方法では、第１の計算機上で実行されている第１のアプリケーションプログラムが発行した複数の送信命令により指定される複数の送信データのそれぞれを、第２の計算機上で実行されている第２のアプリケーションプログラムが発行した複数の受信命令に応答して、メッセージパッシング型の通信で受信する。この受信を第２の計算機上に設けられたエミュレーションライブラリにより制御する。

【００１４】さらに、上記複数の送信データの連なりからなるひと繋がりのデータの内、上記複数の受信命令により指定されるサイズ部分に区分して得られる部分をそれぞれの受信命令が指定する複数のバッファに格納するというストリーム通信を実現するように、受信されたそれぞれの送信データを処理する。この処理を、上記エミュレーションライブラリにより制御する。

【００１５】より具体的には、本発明による計算機間データ送受信方法は、以下の処理を実行する。

【００１６】（ａ）送信側アプリケーションが一回の送信命令によって送信しようとするデータの長さを、受信側エミュレーションライブラリにより検知し、（ｂ）上記の送信データ長を、アプリケーションが一回の受信命令によって指定しているデータ受け取り長と受信側エミュレーションライブラリにより比較し、（ｃ）上記の比較で、送信データ長がデータ受け取り長より長ければ、受信側エミュレーションライブラリが、一旦メモリ上に確保したバッファ領域にデータの全てを受信して、そこからアプリケーションが指定したデータ受け取り長分だけのデータをアプリケーション領域にコピーし、送信データ長がデータ受け取り長以下であれば、受信側エミュレーションライブラリが、アプリケーション領域にデータを直接受信する。

【００１７】（ｄ）アプリケーションが受信命令を発行した時に、バッファ領域にデータが残っている場合には、そこからアプリケーション領域にデータをコピーする。

【００１８】さらにより具体的には、本発明によるデータ送受信方法を実現するために、ＴＣＰ／ＩＰエミュレーションライブラリを用意する。このライブラリは、ＴＣＰ／ＩＰのソケットアプリケーションプログラムインタフェースと同一のインタフェースを持ち、通信相手が並列計算機の外部であったら従来のシステムコールを、内部であったらＭＰＩ等の並列計算機用メッセージパッシング型通信方式による高速通信網を用いる、という切り分けを行う。すなわち、このＴＣＰ／ＩＰエミュレーションライブラリには以下のような特徴を持たせる。

【００１９】（１）並列計算機内部で通信する場合は、メッセージパッシング型通信方式に適した通信手順を用いて、ＴＣＰ／ＩＰと同等のストリーム通信サービスを提供する。

【００２０】（２）ＴＣＰ／ＩＰエミュレーションライブラリとして適当な手段を用いて通信方法を切り分ける。

【００２１】（３）外部通信のデータと内部通信のデータをスピンループによって検出する。もしくは外部通信のデータと内部通信のデータを別々のスレッドで検出する。

【００２２】

【発明の実施の形態】＜従来の技術とその問題点＞本発明の実施の形態を説明する前に、従来の技術とその問題点を説明する。

【００２３】（１）ＴＣＰ／ＩＰ

図１３にＴＣＰ／ＩＰの階層を示す。ＴＣＰ／ＩＰでは、下からリンク層３０１，ＩＰ層３０２，ＴＣＰ層３０３，アプリケーション層３０６の４階層を定めている。実際にはリンク層３０１の下に物理層が存在するが簡単化のためにこの層には言及しない。以下では、簡単化のためにアプリケーション層より下の層３０３，３０

２，３０１からなる層群３０４をまとめてＴＣＰ／ＩＰ層と呼ぶ。一般にＴＣＰ／ＩＰが定める手順で通信を行なうために必要なプログラム（以下、ＴＣＰ／ＩＰ処理ルーチンと呼ぶ）はＯＳに含まれる。ＴＣＰ／ＩＰが定める手順で通信を実行するには、ＴＣＰ／ＩＰ層３０４を構成する複数の層の各々によりそれぞれ特定の処理が実行される。本明細書では、これらの層がそれぞれ実行する複数の処理あるいはそれらの処理を実行する複数のプログラムルーチンを総称してＴＣＰ／ＩＰ処理ルーチンと呼ぶ。

【００２４】アプリケーションプログラムがＴＣＰ／ＩＰ処理ルーチンを利用するためのインタフェースとしては、一般にソケットアプリケーションプログラミングインタフェース（ソケットＡＰＩ）１１１が用いられる。アプリケーションプログラムがＴＣＰ／ＩＰ処理ルーチンを利用するために、ソケットライブラリと呼ばれるシステムコール群がＯＳの一部として設けられている。

【００２５】なお、本明細書では、データ受信のためのシステムコールおよび関数あるいはデータ送信のためのシステムコールおよび関数を呼び出すことを、それぞれの機能と組み合わせて受信命令あるいは送信命令を発行すると記載することがある。ソケットアプリケーションプログラムインタフェースは、ソケットライブラリのアプリケーションプログラムインタフェースとして定められている。ソケットアプリケーションプログラムインタフェースに従って記述された、ＴＣＰ／ＩＰで通信を行なうアプリケーションプログラム（以下、ＴＣＰ／ＩＰアプリケーションプログラムと呼ぶ）の例を以下に示す。

【００２６】

（サーバ側）

```
sa=socket(AF_INET, SOCK_STREAM, 0);
bind(sa, server, slen);
listen(sa, 5);
sa1=accept(sa, &client, &clen);
read(sa1, buffer1, length1);
read(sa1, buffer2, length2);
        .
        .
        .
close(sa1);
close(sa);
```

（クライアント側）

```
sb0=socket(AF_INET, SOCK_STREAM, 0);
connect(sb0, server, slen);
write(sb0, buffer0, length0);
write(sb0, buffer3, length3);
    .
    .
    .
close(sb0);
```

まず、他と通信を行おうとするサーバ側のＴＣＰ／ＩＰアプリケーションプログラムおよびクライアント側のＴＣＰ／ＩＰアプリケーションプログラムは、それが実行中の計算機に設けられたＴＣＰ／ＩＰ処理ルーチンに含まれたシステムコールソケット（ｓｏｃｋｅｔ）を呼び出す。呼び出されたシステムコールソケットは、通信端の役目を果たす、ソケットと呼ばれるオブジェクトを生成し、ソケット記述子を返す。上の例ではサーバ側、クライアント側のＴＣＰ／ＩＰアプリケーションプログラムは、それぞれに返されるソケット記述子をｓａまたはｓｂ０として受け取る。

【００２７】ソケット記述子は、アプリケーションプログラム内でソケット毎に一意に決定されるソケットの識別子（ＩＤ）であり、整数値からなる。生成されたソケットに対する操作は、全てこのソケット記述子を指定して行う。以下、特に断らない限り、ソケットとソケット記述子は同義であるとして説明する。上のシステムコールの引数ＡＦ＿ＩＮＥＴは、アドレスファミリインタネットを表し、インタネットを介した通信にソケットを使用することを示す。さらに、引数ＳＯＣＫ＿ＳＴＲＥＡＭは、ストリーム通信を要求する。最後の引数はプロトコルを指定する引数である。この値が０のときには、プロトコルはその引数より前の二つの引数により決まる。今の場合には、ＴＣＰ／ＩＰプロトコルを使用することになる。ソケットが生成された後、サーバ側とクライアント側のＴＣＰ／ＩＰアプリケーションプログラムは異なる手順でソケットの接続処理を行う。

【００２８】サーバ側は、ＴＣＰ／ＩＰ処理ルーチンに含まれたシステムコールバインド（ｂｉｎｄ）を呼び出す。呼び出されたシステムコールバインドは、引数で指定されたソケットｓａに引数で指定された名前を対応付ける。名前は、ＩＰアドレスとポート番号の組合わせからなる。上の例では、ｓｌｅｎで指定される大きさを持つ構造体ｓｅｒｖｅｒに格納されたＩＰアドレスとポート番号の組からなる名前にソケット識別子ｓａが対応付けられる。サーバ側は、さらにＴＣＰ／ＩＰ処理ルーチンに含まれたシステムコールリスン（ｌｉｓｔｅｎ）を呼び出す。

【００２９】このシステムコールリスンは、引数で指定されたソケットｓａを接続要求の受領のためのソケットに設定する。このシステムコールの第２の引数は、その接続要求および他の接続要求を一時的に保持するのに用いるキューに要求するサイズを表し、今の場合には、このコールは、５個の接続要求を保持可能なキューを要求

している。

【００３０】以上の処理の結果、論理的には、サーバ側とクライアント側の間に通信路が決定されたことになる。

【００３１】その後、サーバ側は、ＴＣＰ／ＩＰ処理ルーチンに含まれたシステムコールアクセプト（ａｃｃｅｐｔ）を呼び出す。このシステムコールは、引数により指定されたソケットｓａを接続要求の待ち状態にする。このシステムコールの第２，第３の引数は、待つべき接続要求の発行元クライアント側のＩＰアドレスと長さを表す。

【００３２】一方、クライアント側は、ＴＣＰ／ＩＰ処理ルーチンに含まれたシステムコールコネクト（ｃｏｎｎｅｃｔ）を呼び出す。このシステムコールは、引数で指定された名前、今の場合にはサーバ側のソケットｓａに付けられた名前（構造体ｓｅｒｖｅｒに格納された、ＩＰアドレスｓｉｎ＿ａｄｄｒとポート番号ｓｉｎ＿ｐｏｒｔの組み合わせ）に対して引数で指定されたソケットｓｂ０を接続する。

【００３３】さらに、サーバ側の計算機では、先に呼び出されたシステムコールアクセプトが、この接続要求を受信し、システムコールアクセプトに対する前述の引数が指定するソケットｓａ１をこのクライアント側との通信のためのソケットとして新たに生成する。こうして、サーバ側のソケットｓａ１とクライアント側のソケットｓｂ０との間で通信路が確立されたことになる。

【００３４】ソケット間の接続が確立された状態でクライアント側がサーバ側にデータを送信する時には、クライアント側は、ＴＣＰ／ＩＰ処理ルーチンに含まれたシステムコールライト（ｗｒｉｔｅ）を呼び出す。このシステムコールに対する引数では、送信すべきデータを保持するのに用いるバッファのアドレスｂｕｆｆｅｒ０とそのバッファの長さｌｅｎｇｔｈ０を指定する。システムコールライトは、このバッファ内のデータをこのシステムコールが指定するソケットｓｂ０を用いて送信する。サーバ側はＴＣＰ／ＩＰ処理ルーチンに含まれたシステムコールリード（ｒｅａｄ）を呼び出す。このシステムコールは、その引数で指定されたソケットｓａ１を介して転送されたデータを受信し、引数で指定されたアドレスｂｕｆｆｅｒ１のバッファに書き込む。このようにして、２つのアプリケーションプログラムの間でデータが転送される。もし必要ならば、クライアント側は複数の後続のデータをそれぞれ送信するために複数のライトシステムコールを呼び出し、サーバ側はそれらの後続のデータを受信するために一つまたは複数のリードシステムコールを呼び出す。

【００３５】ソケット記述子は、ファイルへの入出力や標準入出力などを行う際に用いられる、ファイル記述子の一種として定義されている。このためソケットアプリケーションプログラムインタフェースでは、ファイル等の入出力を行う場合と同様のインタフェースを通して、データの送受信を行うことができる。

【００３６】サーバ側とクライアント側のプログラムの間の通信が終了したときには、サーバ側は、システムコールクローズ（ｃｌｏｓｅ）を呼び出し、引数で指定されたソケットｓａ１を閉鎖する。サーバ側は、さらに同じシステムコールクローズを再度呼び出し、システムコールクローズは今度はソケットｓａを閉鎖する。クライアント側も同様にシステムコールクローズを呼び出して、このシステムコールはソケットｓｂ０を閉鎖する。

【００３７】（２） 高速ソケットにおける通信方法の切り替え

従来のＴＣＰ／ＩＰでは、システムコールバインドによってＩＰアドレスとポート番号からなる名前（ｓｅｒｖｅｒ）をソケット（ｓａ）に対応付けていた。高速ソケットでは、この時、さらに高速通信専用のソケットを以下のように生成する。まず、アプリケーションがシステムコールバインドの呼び出し時に指定する名前のポート番号（構造体ｓｅｒｖｅｒに格納されたポート番号ｓｉｎ＿ｐｏｒｔ）にハッシュ関数を施して、新たにシャドウポート番号を導き出す。次に、システムコールソケットを呼び出して、シャドウソケットを新規に生成する。最後にバインドシステムコールを呼び出して、シャドウソケットにシャドウポート番号を対応付ける。高速通信を行なう場合は、サーバおよびクライアントがａｃｃｅｐｔおよびｃｏｎｎｅｃｔを呼び出して接続するときに、このシャドウソケットを用いることで高速通信専用の通信路を貼る。

【００３８】この方法では、システムコールバインドやシステムコールリスンが呼び出される時に、特別な処理が必要となる。

【００３９】（3） ＴＣＰ／ＩＰ処理ルーチンによるストリーム通信

ＴＣＰ／ＩＰのソケットライブラリは、ストリーム通信をサービスする。ストリーム通信とは、送信側のアプリケーションプログラムが一連のライトシステムコールにより送る複数のデータをひと繋がりのデータストリームとして処理し、そのデータストリームを受信側のアプリケーションプログラムが発行する一つあるいは複数のリードシステムコールが指定する任意の長さを有する一つあるいは複数のデータに切り分けて受け取るという通信方法である。

【００４０】図１４を用いて従来のストリーム通信の動作を説明する。ここでは、送信アプリケーションプログラム８０１が受信アプリケーションプログラム８０２にデータを送る例を示している。送信アプリケーションプログラム８０１は、まず第１のライトシステムコールを呼び出してそのプログラムのバッファ８０３内の５０キロバイト（ＫＢ）のデータを送信する（８０５）。なお、図１４では単位ＫＢは簡単化のために省略してい

る。送信側のＴＣＰ／ＩＰ処理ルーチンは、バッファ８０３内のこの送信データを一旦ＯＳ内に確保したバッファ（図示せず）にコピーし、複数のパケットに分割して、受信側ＯＳに送信する。 通常パケットのサイズは４０～１５００バイトである。受信側ＯＳは、それらのパケットをＯＳ内に確保した複数のバッファ（図示せず）に受け取り、これらのパケットをリスト状に繋いで、データストリームを再構成する。送信アプリケーションプログラム８０１は、さらに第２のライトシステムコールを発行してそのプログラムの他のバッファ８０４内の８０ＫＢのデータを送信する（８０６）。このときも送信側のＴＣＰ／ＩＰ処理ルーチンと受信側のＴＣＰ／ＩＰ処理ルーチンは同様に動作して、このデータを先に送信されたデータと組み合わせて一つのデータストリームとしてＯＳ内の前述のバッファに保持する。

【００４１】図において、８０９はＯＳ９００内に保持されたこのデータストリームを模式的に表す。先頭のデータ８０７、後続のデータ８０８はそれぞれ上記第１、第２のライトシステムコールにより送信されたそれぞれ５０ＫＢ、８０ＫＢのデータを表す。ストリーム通信では、受信アプリケーションプログラムは、これらのデータを一本の連続した１３０ＫＢのデータストリーム８０９として捉える。送信アプリケーションプログラム８０１内のバッファ８０３、８０４内のデータは、ＯＳ９００内のデータストリーム８０９内のデータ部分８０７，８０９をそれぞれ保持する。このための、受信アプリケーションプログラム８０２は、第１のリードシステムコールを発行して、ストリームデータ８０９内の５０ＫＢの先頭データ８０７の内の３０ＫＢを受信アプリケーションプログラム８０２の３０ＫＢのバッファ８１２に受け取ることを要求する（８１４）。受信側のＴＣＰ／ＩＰ処理ルーチンは、このリードシステムコールに応答して、このデータストリーム８０９から先頭の３０ＫＢのデータ８１０をバッファ８１２にコピーする。受信アプリケーションプログラム８０２がさらに第２のリードシステムコールを発行すると、このシステムコールが指定する長さに従って、ストリームデータ８０９内にある残りの１００ＫＢのデータ８１１をこのシステムコールが指定するバッファ８１３にコピーする（８１５）。

【００４２】（4）アクティブメッセージ型通信でのストリーム通信

上記高速ソケットを使用したシステムでは、アクティブメッセージ型の通信を採用しながらストリーム通信を実現している。すなわち、このシステムでは、送信側のアプリケーションプログラムから送信要求が発行されると、この送信要求は、受信側のアプリケーションプログラムから受信要求が発行されるのを待たないで実行される。送信要求は、受信側の計算機に割り込みを発生し、この割り込みにより割り込みハンドラーが起動され、送信データは、この割り込みハンドラー内のバッファに一旦受信される。受信側のアプリケーションプログラムが受信要求を発行すると、割り込みハンドラーは、既に受信されたデータの内、この受信要求が要求する大きさのデータを、受信側のアプリケーションプログラムのバッファに転送する。もし、受信側のアプリケーションプログラムが要求するサイズが、受信済みのデータのサイズより小さければ、受信要求の処理が終了する。割り込みハンドラーに保持された残りのデータは、受信側のアプリケーションプログラムから新たな受信要求が発行されたときに、そのアプリケーションプログラムのバッファに転送される。

【００４３】逆に、受信側のアプリケーションプログラムが要求するサイズが、受信済みのデータのサイズより大きければ、割り込みハンドラーは、送信側のアプリケーションプログラムがその後データを送信してきたときに、そのデータを、受信側のアプリケーションプログラムに供給する。受信側の割り込みハンドラーは、送信側のアプリケーションプログラムが送信要求を発行する前に、受信側のアプリケーションプログラムから後続の受信要求が発行されたときにも、同様にその後送信側のアプリケーションプログラムが後続のデータをその後送信してきたときに、その後続のデータを受信側のアプリケーションプログラムに供給する。もし、この後続のデータが、上記最初の受信要求が要求する不足のデータのサイズあるいは上記後続の受信要求が要求するデータのサイズより大きければ、割り込みハンドラーは、あまりのデータをさらに後続の受信要求のために保持する。

【００４４】こうして、送信側のアプリケーションプログラムが発行する複数の送信要求により送信される複数のデータを、受信側のアプリケーションプログラムが発行する一連の受信要求に応答して受信側のアプリケーションプログラムに供給する。こうして、この方法では、割り込みハンドラー内のバッファに送信データを一旦保持することにより、ストリーム通信を実現している。

【００４５】しかし、割込を使用しないメッセージパッシング型の通信では、このような方法を使用してストリーム通信を実現できない。

【００４６】以下、本発明に係る計算機間データ送受信方法を図面に示したいくつかの実施の形態を参照してさらに詳細に説明する。なお、以下においては、同じ参照番号は同じものもしくは類似のものを表わすものとする。また、発明の第２の実施の形態以降においては、発明の第１の実施の形態との相違点を主に説明するに止める。

【００４７】＜発明の実施の形態１＞

（１）装置の概要

図１は、本発明に係る計算機間送受信方法を実行するための計算機システムの一例を示す。図において、並列計算機１０１の内部の２台の計算機１０２，１０３と、１台の外部計算機１０４とがお互いに通信網で繋がってい

ると仮定する。実際には、並列計算機１０１の内部および外部の計算機の台数はそれぞれ任意である。内部の計算機１０２と１０３は内部高速通信網１０５で繋がっており、内部の計算機１０２，１０３にはそれぞれ内部高速通信網１０５専用のネットワークインタフェースハードウェア１１９，１２０が存在する。内部高速通信網１０５は、複数のパケットを互いに並列にかつ高速に転送可能なネットワーク、たとえばハイパクロスバスイッチなどにより構成される。また、内部の計算機１０２，１０３の全てと外部の計算機１０４はグローバルな通信網１０６に繋がっており、各計算機にはそれぞれ通信網１０６専用のネットワークインタフェースハードウェア１２１，１２２，１２３が存在する。

【００４８】データの送受信は、各計算機のメモリ１２４，１２５，１２６にロードされているアプリケーションプログラム１０７，１０８，１０９の間で行われる。また、各メモリにはアプリケーションプログラムの他にＯＳ１２７，１２８，１２９がロードされており、それぞれのＯＳの中にはＴＣＰ／ＩＰ処理ルーチン１１４，１１５，１１０が存在する。ＴＣＰ／ＩＰが定める手順で通信を実行するには、ＴＣＰ／ＩＰ層３０４を構成する各層によりそれぞれ処理が実行される。ＴＣＰ／ＩＰ処理ルーチンは、ＴＣＰ／ＩＰが定める手順で通信を実行するためにこれらの層がそれぞれ実行する複数の処理の総称で、これらのＴＣＰ／ＩＰ処理ルーチン１１４，１１５，１１０自体は公知のものと同じであり、既に述べたような、システムコールで呼び出し可能な複数の関数を含んでいる。ＴＣＰ／ＩＰ処理ルーチン１１４，１１５，１１０は、広域通信を目的として、グローバルな通信網１０６専用のネットワークインタフェースハードウェア１２１，１２２，１２３によって通信を行う。

【００４９】計算機１０２のメモリ１２４には、さらにＴＣＰ／ＩＰエミュレーションライブラリ１１２とメッセージパッシング型ライブラリ１４０と高速通信ライブラリ１３５とがロードされている。同様に、計算機１０３のメモリ１２５には、ＴＣＰ／ＩＰエミュレーションライブラリ１１３とメッセージパッシング型ライブラリ１４１と高速通信ライブラリ１３６とがロードされている。

【００５０】計算機１０２，１０３上の高速通信ライブラリ１３５，１３６は並列計算機１０１の内部での高速通信を目的として、内部高速通信網１０５専用のネットワークインタフェースハードウェア１１９，１２０によって通信を行うためのライブラリであり、リモートメモリ書き込みライブラリあるいはＰＵＴライブラリと呼ばれているものが多く使用される。本実施の形態でも高速通信ライブラリ１３５、１３６にはこのＰＵＴライブラリを使用する。しかし、本発明はこのライブラリに限定されるのではなく、他のライブラリたとえばＰＵＴ／ＧＥＴライブラリと呼ばれるライブラリも使用可能である。

【００５１】一般に、高速内部通信網１０５を通した通信はグローバルな通信網１０６を通して通信する場合に比べて格段に速い。そこでＴＣＰ／ＩＰエミュレーションライブラリ１１２，１１３は、内部計算機上のアプリケーションプログラム同士がＴＣＰ／ＩＰ通信を行おうとする際には、ＴＣＰ／ＩＰ処理ルーチン１１４，１１５ではなくメッセージパッシング型ライブラリ１４０，１４１と高速通信ライブラリ１３５，１３６と内部高速通信網１０５を使用してメッセージパッシング型の通信を実現するように構成され、それでもって通信の高速化を図る。

【００５２】メッセージパッシング型ライブラリ１４０、１４１は、ＴＣＰ／ＩＰエミュレーションライブラリ１１２または１１３からの要求にしたがって、高速通信ライブラリ１３５または１３６を起動するためのライブラリである。メッセージパッシング型ライブラリ１４０、１４１は、一般的にはアプリケーションプログラム（本実施の形態においてはＴＣＰ／ＩＰエミュレーションライブラリ１１２、１１３）に対してメッセージパッシング型のインタフェースを有するライブラリである。

【００５３】ＴＣＰ／ＩＰエミュレーションライブラリ１１２，１１３は、さらに、このメッセージパッシング型の通信においても従来のＴＣＰ／ＩＰ処理ルーチンが提供していたのと同じくストリーム通信を実現し、それでもって通信の高速化を図る。ＴＣＰ／ＩＰ処理ルーチン１１４，１１５は、ＯＳ１２７，１２８の機能であるため、従来技術ではアプリケーションプログラムがこれらを利用する際には必ずコンテクストスイッチのオーバーヘッドが発生するが、本実施の形態では、高速通信ライブラリ１３５，１３６を使うので、ＯＳを介さないためオーバーヘッドを回避でき、それにより、より一層の通信の高速化も期待できる。

【００５４】

（２）論理構成と構成要素間のインタフェース図１（Ｂ）は、上記ハードウェア構成を論理構成として表現した図である。この図では、以降の説明に関係の無い部分は全て省略してある。また、高速通信ライブラリ１３５，１３６と内部高速通信専用ハードウェア１１９，１２０をひとまとめにして高速通信機構１１６，１１７と表現している。さらに、各構成要素間のインタフェース１１１および１１８を新たに示している。計算機１０２，１０３，１０４や並列計算機１０１を表す四角はそれぞれ、その中の論理構成要素が一台の計算機または並列計算機１０１上で実行されることを示している。

【００５５】計算機１０４上では従来通り、アプリケーションプログラム１０９がＴＣＰ／ＩＰ１１０と、ソケットアプリケーションプログラムインタフェース１１１でリンクされている。並列計算機１０１の内部の計算機１０２，１０３上で動くアプリケーションプログラム１

０７，１０８は、ＴＣＰ／ＩＰエミュレーションライブラリ１１２，１１３にソケットアプリケーションプログラムインタフェース１１１でもってリンクされている。また、ＴＣＰ／ＩＰエミュレーションライブラリ１１２，１１３は、ＯＳの機能である従来のＴＣＰ／ＩＰ処理ルーチン１１４，１１５にソケットアプリケーションプログラムインタフェース１１１でもってリンクされ、同時に、高速通信機構１１６，１１７にＭＰＩ仕様のインタフェース１１８でもってリンクされている。

【００５６】（３）アプリケーションプログラム１０７，１０８

本実施の形態では、並列計算機１０１内のいずれかの計算機１０２上で動作しているアプリケーションプログラム例えば１０７が、いずれかの他の計算機上で動作している他のアプリケーションプログラムと通信する場合、当該他のアプリケーションプログラムが、並列計算機１０１内の計算機１０３上で動作しているアプリケーションプログラム例えば１０８か並列計算機１０１外の計算機１０４上で動作しているアプリケーションプログラム例えば１０９かによって、高速通信機構１１６とＴＣＰ／ＩＰ処理ルーチン１１４を使い分ける。並列計算機１０１内の異なる計算機１０２、１０３上で動作する二つのアプリケーションプログラム１０７、１０８が高速通信機構１１６、１１７を使用して相互に通信するためには、それらのアプリケーションプログラム間でデータを実際に送受信するときだけでなく、それぞれのアプリケーションプログラムのためのソケットを相互に接続するときにも高速通信機構１１６、１１７を利用するように特別の処理を行う必要がある。ＴＣＰ／ＩＰエミュレーションライブラリ１１２または１１３がこの特別の処理を実行するための複数の関数を含む。本実施の形態では、ＴＣＰ／ＩＰエミュレーションライブラリ１１２、１１３内に設けられた関数の名前にＥＭＵ＿という接頭辞をつけ、ＴＣＰ／ＩＰ処理ルーチン１１４、１１５または１１０内に設けられた前述の関数の名前と区別する。

【００５７】具体的には、並列計算機１０１内の計算機１０２、１０３上で動作しているアプリケーションプログラム１０７または１０８の内、サーバ側およびクライアント側として動作するアプリケーションプログラムはそれぞれ以下のプログラムを実行するように生成される。

【００５８】

```
（サーバ側）                                    （クライアント側）
sa=socket(AF_INET, SOCK_STREAM, 0);
                                        sb0=socket(AF_INET, SOCK_STREAM, 0);
bind(sa, server, slen);
listen(sa, 5);
sal=EMU_accept(sa, &client, &clen);     EMU_connect(sb0, server, slen);
EMU_read(sal, buffer1, length1);        EMU_write(sb0, buffer0, length0);
EMU_read(sal, buffer1, length1);        EMU_write(sb0, buffer0, length0);
                .                               .
                .                               .
                .                               .
close(sal);
close(sa);                                      close(sb0);
```

サーバ側アプリケーションプログラムは従来と同様にｓｏｃｋｅｔ、ｂｉｎｄ、ｌｉｓｔｅｎに対するシステムコールを呼び出す。これらのシステムコールは対応するＴＣＰ／ＩＰ処理ルーチンにより従来と同様に処理される。クライアント側のアプリケーションプログラムも従来と同様にｓｏｃｋｅｔシステムコールを発行する。このシステムコールも対応するＴＣＰ／ＩＰ処理ルーチンにより従来と同様に処理される。こうしてサーバ側とクライアント側に対して従来と同様にソケットｓａ、ｓｂ０が生成される。

【００５９】その後サーバ側アプリケーションプログラムは、従来のシステムコールアクセプトに代えて、そのアプリケーションプログラムが動作している計算機内に設けられたＴＣＰ／ＩＰエミュレーションライブラリ内に設けられた関数エミュレーションアクセプト（ＥＭＵ＿ａｃｃｅｐｔ）を呼び出す。クライアント側のアプリケーションプログラムは、従来の関数コネクトに代えて、そのアプリケーションプログラムが動作している計算機内に設けられたＴＣＰ／ＩＰエミュレーションライブラリ内に設けられた関数エミュレーションコネクト（ＥＭＵ＿ｃｏｎｎｅｃｔ）を呼び出す。さらに、サーバ側アプリケーションプログラムは、従来のシステムコールリードに代えて、ＴＣＰ／ＩＰエミュレーションライブラリ内に設けられた関数エミュレーションリード（ＥＭＵ＿ｒｅａｄ）を呼び出し、クライアント側アプリケーションプログラムは、従来のシステムコールライトに代えて、対応するＴＣＰ／ＩＰエミュレーションライブラリ内に設けられた関数エミュレーションライト（ＥＭＵ＿ｗｒｉｔｅ）を呼び出す。以下、これらの新たな関数が行う処理を説明する。

【００６０】（４）ＴＣＰ／ＩＰエミュレーションライブラリによるソケットの接続
図２において、サーバ側アプリケーションプログラム，クライアント側アプリケーションプログラムが、それぞれ上述のＥＭＵ＿ａｃｃｅｐｔ，ＥＭＵ＿ｃｏｎｎｅｃｔ関数を呼び出すと（処理５０１，５０２）、これらのシステムコールによってサーバ側のＴＣＰ／ＩＰエミュレーションライブラリ内に設けられた関数ＥＭＵ＿ａｃｃｅｐｔとクライアント側のＴＣＰ／ＩＰエミュレーションライブラリ内に設けられたＥＭＵ＿ｃｏｎｎｅｃｔ呼び出される。これらのシステムコールの引数は、サーバ側アプリケーションプログラム，クライアント側アプリケーションプログラムが、それぞれシステムコールａｃｅｐｔ、ｃｏｎｎｅｃｔを呼び出し、ソケットｓａ、ｓｂ０の接続をＴＣＰ／ＩＰ処理ルーチン１１４，１１５に要求するときと同じである。
【００６１】呼び出された関数ＥＭＵ＿ａｃｃｅｐｔとＥＭＵ＿ｃｏｎｎｅｃｔは、まずａｃｃｅｐｔシステムコール，ｃｏｎｎｅｃｔシステムコールをそれぞれ発行する（処理５０３，５０４）。これらのシステムコールの引数は、関数ＥＭＵ＿ａｃｃｅｐｔ，ＥＭＵ＿ｃｏｎｎｅｃｔのそれぞれに対する引数がそのまま使用される。これによりサーバ側のＴＣＰ／ＩＰ処理ルーチン内に設けられたシステムコールａｃｃｅｐｔとクライアント側のＴＣＰ／ＩＰ処理ルーチン内に設けられたシステムコールｃｏｎｎｅｃｔが呼び出され、従来と同様にコールされたシステムコールａｃｃｅｐｔはコールされたシステムコールｃｏｎｎｅｃｔによって発行された接続要求を受領し、ソケットｓａ１が生成され、ソケットｓａ１とソケットｓｂ０が通信路１０６を介して接続された状態になる。
【００６２】その後、関数ＥＭＵ＿ａｃｃｅｐｔ，ＥＭＵ＿ｃｏｎｎｅｃｔは、それぞれ相手のＩＰアドレスが並列計算機１０１の内部の計算機のアドレスであるか否かを確認する（処理５０６，５０７）。相手が並列計算機１０１の内部である場合は、ソケット記述子を内部テーブル４１０または４１１に登録する（処理５１０，５１１）。今の場合には、サーバ側のアプリケーションプログラム１０７用のソケットｓａ１、クライアント側のアプリケーションプログラム用のソケットｓｂ０がそれぞれ内部テーブル４１０、４１１にそれぞれ登録される。さらに、ＴＣＰ／ＩＰエミュレーションライブラリ１１２、１１３は、並列計算機１０１内における各計算機の識別子などの高速通信機構１１６，１１７を用いるために必要なデータを交換する。このデータ交換のために、アプリケーションプログラム１０７、１０８はそれぞれ関数ｒｅａｄ，ｗｒｉｔｅシステムコールを発行する（処理５１２、５１３、５１５、５１６）。ここでのデータ交換は、既に接続されているソケットｓａ１とｓｂ０を利用し、従来と同様にＴＣＰ／ＩＰ処理ルーチン１１４，１１５と通信路１０６を介して行われる。その後それぞれサーバ側、クライアント側のアプリケーションプログラム１０７，１０８にリターンする（処理５１８，５１９）。このとき、サーバ側のＴＣＰ／ＩＰエミュレーションライブラリ１１２はアプリケーションプログラム１０７に生成されたソケットｓａ１の識別子を戻す。
【００６３】処理５０６，５０７において、相手が並列計算機１０１の内部の計算機でないことが判明したときは、関数ＥＭＵ＿ｃｏｎｎｅｃｔはクライアント側のアプリケーションプログラム１０８に返り、関数ＥＭＵ＿ａｃｃｅｐｔはソケットｓａ１をサーバ側のアプリケーションプログラム１０７に返し、そのプログラムに戻る（処理５０８，５０９）。こうして、サーバ側のアプリケーションプログラムに対するソケットｓａ１とクライアント側のアプリケーションプログラムに対するソケットｓｂ０は、それぞれに対応するＴＣＰ／ＩＰ処理ルーチンと通信路１０６を介して接続される。
【００６４】なお、図３は、本実施の形態によりアプリケーションプログラム１０７，１０８，１０９が生成したソケットがお互いに接続されている一つの状態を示している。ここでは、アプリケーションプログラム１０７のソケットＳＡ１（４０２）とアプリケーションプログラム１０８のソケットＳＢ０（４０３）とが接続され（４０７）、アプリケーションプログラム１０８のソケットＳＢ１（４０４）とアプリケーションプログラム１０９のソケットＳＣ０（４０５）が接続され（４０８）、アプリケーションプログラム１０９のソケットＳＣ１（４０６）とアプリケーションプログラム１０７のソケットＳＡ０（４０１）が接続されている（４０９）。ここで、ＴＣＰ／ＩＰエミュレーションライブラリ１１２，１１３は、内部ソケットテーブル４１０，４１１を保持している。内部ソケットテーブ４１０、４１１には、ソケットの接続先が内部計算機である場合に、そのソケット記述子を登録する。例えば、アプリケーションプログラム１０７のソケットＳＡ１（４０２）の接続先であるソケットＳＢ０（４０３）は、内部計算機１０３の上で動くアプリケーションプログラム１０８のソケットなので、ＳＡ１を内部ソケットテーブル４１０に登録する。同様に、ソケットＳＢ０の接続先であるソケットＳＡ１は、内部計算機１０２の上で動くアプリケーションプログラム１０７のソケットなので、ＳＢ０を内部ソケットテーブル４１１に登録する。一方、ソケットＳＡ０（４０１）の接続先であるソケットＳＣ１（４０６）は、外部計算機上アプリケーションプログラム１０９のソケットなので、ＳＡ０は内部ソケットテーブル４１０には登録しない。同様に、ＳＣ０に接続されているＳＢ１も内部ソケットテーブル４１１には登録しない。
【００６５】（５）内部通信と外部通信の切り分け方法
その後、サーバ側のアプリケーションプログラムとクラ

イアント側のアプリケーションプログラムはデータ通信を開始する。これらの二つのプログラムの内の一方および他方は、データ送信および受信のために関数ＥＭＵ＿ｗｒｉｔｅ，ＥＭＵ＿ｒｅａｄをそれぞれ呼び出す。先に示したクライアント側とサーバ側のプログラムの例では、サーバ側のアプリケーションプログラムが、関数ＥＭＵ＿ｒｅａｄを呼び出し、クライアント側のアプリケーションプログラムが関数ＥＭＵ＿ｗｒｉｔｅを呼び出している。これらで指定する引数は、ＴＣＰ／ＩＰ処理ルーチン１１４，１１５に含まれたシステムコールｗｒｉｔｅ、ｒｅａｄに対する引数と同じである。

【００６６】図４を参照するに、ＥＭＵ＿ｒｅａｄ，ＥＭＵ＿ｗｒｉｔｅが呼び出されると（処理７０１、７０２）、それぞれの関数は、それぞれの引数で指定されたソケットｓａ１、ｓｂ０のソケット記述子が対応する内部ソケットテーブル４１０、４１１にそれぞれ登録されているか否かを判定する（処理７０３，７０４）。それぞれのソケット識別子が内部ソケットテーブル４１０，４１１にそれぞれ登録されているときには、後に詳細に説明する手順で高速通信機構１１６，１１７、高速内部通信網１０５を用いてメッセージパッシング方式の通信を行う（処理７０７，７０８）。それぞれのソケット識別子が内部ソケットテーブル４１０、４１１に登録されていなかったら、そのまま従来のｒｅａｄ，ｗｒｉｔｅシステムコールをそれぞれ指定されたソケットに対して発行する（処理７０５，７０６）。これらのシステムコールは対応するＴＣＰ／ＩＰ処理ルーチン１１４，１１５により処理され、グローバル通信路１０６を用いた通信がそれらのＴＣＰ／ＩＰ処理ルーチン１１４，１１５により実行される。

【００６７】なお、図５は、図３で示したソケットの接続状態において、各アプリケーションプログラム１０７，１０８，１０９同士が通信を行う場合のデータの流れを示して、上記切り分け方法を説明するための全体構成図である。アプリケーションプログラム１０７とアプリケーションプログラム１０８が通信する場合は、ソケットｓａ１（４０２）とソケットｓｂ０（４０３）を通信端として用いる。これらのソケットは、ＴＣＰ／ＩＰエミュレーションライブラリ１１２，１１３が保持する内部ソケットテーブル４１０，４１１に登録されている。そこでＴＣＰ／ＩＰエミュレーションライブラリ１１２，１１３は、データ通信処理にＴＣＰ／ＩＰ処理ルーチン１１４，１１５ではなく高速通信機構１１６，１１７を利用する（６０１）。一方、アプリケーションプログラム１０８とアプリケーションプログラム１０９が通信する場合は、ソケットＳＢ１（４０４）とソケットＳＣ０（４０５）を通信端として用いる。アプリケーションプログラム１０８とリンクされているＴＣＰ／ＩＰエミュレーションライブラリ１１３が保持する内部ソケットテーブル４１１には、ＳＢ１は登録されていない。そこでＴＣＰ／ＩＰエミュレーションライブラリ１１３は、データ通信処理に、ＴＣＰ／ＩＰ処理ルーチン１１５をそのまま利用する（６０２）。これによって、外部のＴＣＰ／ＩＰ処理ルーチン１１０とのデータ交換が可能となる。

【００６８】本方式により、バインドシステムコール、リスンシステムコールは従来のＴＣＰ／ＩＰ処理ルーチンから変更せずにそのまま用いて、内部通信と外部通信の切り分けを実現できる。

【００６９】（６）メッセージパッシング型ライブラリ１４０，１４１

並列計算機の各計算機に使用される内部高速通信網１０５を使用するための通信ハードウェア１１９、１２０および高速通信ライブラリ１３５，１３６はベンダ特有である場合が多いので、その利用方法もマシンによって様々である。よって、各マシンの高速通信ハードウェアと高速通信ライブラリを利用したアプリケーションプログラムを作ろうとする場合、マシンに特化した汎用性の低いプログラムにならざるを得なかった。これに対して、並列計算機内部の通信ハードウェアを利用して通信するためのライブラリを用意し、このライブラリのアプリケーションプログラムインタフェースを標準として規定することで、アプリケーションプログラムの汎用性を高めようという活動が世界中で活発である。

【００７０】このメッセージ型の通信を使用するための汎用のインタフェースとして現在広く使用されているインタフェースは、ＭＰＩと呼ばれるメッセージパッシングインタフェース（ＭＰＩ－Ｍｅｓｓａｇｅ　Ｐａｓｓｉｎｇ　Ｉｎｔｅｒｆａｃｅ）である。例えば、文献："MPI: Message Passing Interface Standard version1.1", MPI Forum, University of Tennessee, 1995参照。このインタフェースは、メッセージパッシング型の通信を実現するための上に述べた高速通信ライブラリが存在することを前提としているものであり、このインタフェースを使用しても、メッセージパッシング型の通信は、基本的には上記高速通信ライブラリにより実現されることには変わらない。

【００７１】多くの並列計算機ベンダが、ＭＰＩに準拠したアプリケーションプログラムが並列計算機内部の高速通信ハードウェアを使用できるようにするための高速通信ライブラリを提供している。

【００７２】本明細書では、メッセージ型の通信を使用するためのインタフェースをメッセージパッシング型インタフェースと呼び、そのインタフェースを有するライブラリをメッセージパッシング型ライブラリと呼ぶ。特に、ＭＰＩ仕様のインタフェースをＭＰＩあるいはＭＰＩインタフェースあるいはメッセージパッシングインタフェースと呼び、そのインタフェースを有するライブラリをＭＰＩライブラリあるいはメッセージパッシングインタフェースライブラリと呼ぶことがある。

【００７３】本実施の形態では、メッセージパッシング型ライブラリ１４０、１４１として多くの並列計算機で利用可能である標準のＭＰＩ仕様により定められたインタフェースでもってコマンドあるいはデータを交換するライブラリを使用する。それでもって、ＴＣＰ／ＩＰエミュレーションライブラリ１１２，１１３の汎用性を高める。しかし、他の仕様のインタフェースを使用してもよい。

【００７４】ＭＰＩに準拠した従来のアプリケーションプログラムの記述例を以下に示す。本実施の形態では、ＴＣＰ／ＩＰエミュレーションライブラリ１１２、１１３は、メッセージパッシング型ライブラリ１４０、１４１を起動する部分に関しては、以下に示すプログラム部分を有する。このプログラム部分のより詳細は、後にふれる。

【００７５】

```
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
if ( rank == 0 )          /*  receiver   */
  MPI_Recv(buffer, length, type, 1, tag, comm, status);
else if ( rank == 1)      /*  sender   */
  MPI_Send(buffer, length, type, 0, tag, comm);
MPI_Finalize();
```

ＭＰＩでは、通信しようとする全プロセスを一斉に立ち上げる。この時、各プロセスにはランクと呼ばれるプロセス識別子が決定される。上の例では、２つのプロセス、すなわち送信側プロセスｓｅｎｄｅｒと受信側プロセスｒｅｃｅｉｖｅｒとを立ち上げる場合を示している。この時、それぞれのプロセスにはランク０と１が付けられる。各プロセスは、まずＭＰＩ初期化関数ＭＰＩ＿Ｉｎｉｔによって ＭＰＩの初期化を行い、ＭＰＩ通信ランク関数ＭＰＩ＿Ｃｏｍｍ＿ｒａｎｋによって自分のランクを取得する。その後、各ランクごとの処理を行う。上の例では、ランク１のプロセスがＭＰＩ送信関数ＭＰＩ＿Ｓｅｎｄによって送るデータを、ランク０のプロセスがＭＰＩ受信関数ＭＰＩ＿Ｒｅｃｖで受け取る場合を示している。

【００７６】（７）高速通信機構を用いたメッセージパッシング型の通信

高速通信機構１１６，１１７を用いるデータ受信処理７０７，データ送信処理７０８の詳細を説明する前に、これらの処理の実行に必要な、高速通信機構１１６，１１７を用いたメッセージパッシング型の通信の概要を説明する。

【００７７】一般にメッセージパッシング型の通信を実行するには、送信側の計算機がデータを送信するための送信関数を実行し、受信側の計算機がそのデータを受信するための受信関数を実行する。後続のデータをそれらの計算機の間で転送するために、それらの関数が繰り返し実行される。本実施の形態では、メッセージパッシング型の通信の具体的な説明として、ＭＰＩ仕様によるメッセージパッシング通信を説明する。

【００７８】ＭＰＩ仕様によるメッセージパッシング通信では、送信側の計算機は送信関数ＭＰＩ＿Ｓｅｎｄを実行し、受信側の計算機は受信関数ＭＰＩ＿Ｒｅｃｖを実行する。今の場合、送信処理７０８が起動されると、送信側のＴＣＰ／ＩＰエミュレーションライブラリ１１３は、先に記載したＭＰＩ仕様のアプリケーションプログラムの例にあるように、送信関数ＭＰＩ＿Ｓｅｎｄを呼び出す。なお、送信関数ＭＰＩ＿Ｓｅｎｄを呼び出す前に初期処理としてＭＰＩ＿Ｉｎｉｔ、ＭＰＩ＿Ｃｏｍｍ＿ｒａｎｋ等の関数の呼び出しを実行する必要があるが、これらの処理は、先に記載した関数ＥＭＵ＿ａｃｃｅｐｔ，ＥＭＵ＿ｃｏｎｎｅｃｔ内の内部テーブル登録処理（５１０，５１１）と同時に行なう。一方、受信側のＴＣＰ／ＩＰエミュレーションライブラリ１１２は、内部高速通信網１０５を用いるために受信関数ＭＰＩ＿Ｒｅｃｖを呼び出す。

【００７９】送信側のＴＣＰ／ＩＰエミュレーションライブラリ１１３が上記送信処理７０８内で呼び出す関数ＭＰＩ＿Ｓｅｎｄの引数で指定するバッファアドレスｂｕｆｆｅｒおよびバッファ長ｌｅｎｇｔｈは、送信側のアプリケーションプログラム１０８が発行した関数ＥＭＵ＿ｗｒｉｔｅの対応する引数に等しくされる。一方、受信側のＴＣＰ／ＩＰエミュレーションライブラリ１１２が受信処理７０７内で呼び出す関数ＭＰＩ＿Ｒｅｃｖの引数で指定するバッファアドレスｂｕｆｆｅｒおよびバッファ長ｌｅｎｇｔｈは、後に説明するように受信側のアプリケーションプログラム１０７が発行した、関数ＥＭＵ＿ｒｅａｄの引数が指定するバッファアドレスおよびバッファ長もしくは受信側のＴＣＰ／ＩＰエミュレーションライブラリ１１３内に設けられる特定のバッファのアドレスおよびサイズに等しくされる。

【００８０】送信側のメッセージパッシング型ライブラリ１４１内の関数ＭＰＩ＿Ｓｅｎｄは、起動されると、対応する高速通信ライブラリ１３６に対し、関数ＭＰＩ＿Ｓｅｎｄに対する引数が指定するアドレスのバッファからその引数が指定する長さのデータを内部高速通信網１０５を介して受信側の計算機１２４に転送することを要求する。受信側のメッセージパッシング型ライブラリ１４０内の関数ＭＰＩ＿Ｒｅｃｖは、起動されると、対応する高速通信ライブラリ１３５に対し、その高速通信ライブラリ１３５に、この転送されたデータを内部高速

通信網１０５を介して受信し、関数ＭＰＩ＿Ｒｅｃｖに対する引数が指定するアドレスのバッファにその引数が指定する長さのデータを書き込むことを要求する。

【００８１】起動された送信側の高速転送ライブラリ１３６と受信側の高速転送ライブラリ１３５は、要求されたデータの送信と受信をそれ自体公知の方法により内部高速通信網１０５を介して行う。このデータ転送は具体的には以下のように行われる。一般にはリモートメモリ書き込みコマンドあるいはＰＵＴコマンドといわれるコマンドが使用される。以下では、このコマンドをＰＵＴコマンドと呼ぶ。内部高速通信網１０５を介したデータ転送を行う通信方法は３つのＰＵＴコマンドにより行われる。

【００８２】まず、送信側のＴＣＰ／ＩＰエミュレーションライブラリ１１３から関数ＭＰＩ＿Ｓｅｎｄが呼び出されると、送信側のメッセージパッシング型ライブラリ１４１は、この引数が指定したバッファ長（これはすなわち送信すべきデータの長さである）を含むデータの属性情報を含むヘッダの送信を送信側の高速転送ライブラリ１３６に要求する第１のＰＵＴコマンドを発行する。この高速転送ライブラリ１３６は、受信側の計算機１０２にこのヘッダを送信する。計算機１０２内の内部高速通信専用ハードウェア１１９が、このデータを受信側のメモリ１２４内の所定の位置に直接書き込む。

【００８３】一方、受信側ＴＣＰ／ＩＰエミュレーションライブラリ１１２から受信命令ＭＰＩ＿Ｒｅｃｖが呼び出されると、受信側のメッセージパッシング型ライブラリ１４０は、まず送信側の計算機１０３からヘッダがすでに送られてきているか調べる。このために後述するデータ検査命令ＭＰＩ＿ｐｒｏｂｅが使用される。ヘッダが送られてきていないときには、その受信命令の実行は終了する。ヘッダがすでに送られてきているときには、そのヘッダ内のバッファ長から送信データが受信可能か否かを判定する。すなわち、その送信データの長さが、上記受信命令ＭＰＩ＿Ｒｅｃｖが指定したバッファ長以下であるか否かが判定される。ＭＰＩ仕様のメッセージパッシング通信に限らず、一般にメッセージパッシング型の通信では、送信命令が指定した送信データ長が、受信命令が指定した受信バッファサイズより大きいときには、受信側の計算機はその送信データを受信しないようになっている。もし受信側の計算機が受信側のバッファのサイズを超える受信データを受信したときには、そのバッファ以外のメモリ領域が受信データにより破壊する恐れがあるためである。受信側のメッセージパッシング型ライブラリ１４０は、上記判定の結果、送信データが受信可能であると判断したときには、受信命令ＭＰＩ＿Ｒｅｃｖに対する引数が指定した受信バッファのアドレスの送信を受信側の高速通信ライブラリ１３５に要求する第２のＰＵＴコマンドを発行する。この高速通信ライブラリ１３５はこのコマンドに対する応答としてバッファアドレスを送信側の計算機１０３に送信する。１０２内の内部高速通信専用ハードウェア１１９は、このアドレスを送信側のメモリ１２５内の所定の位置に書き込む。なお、上記判定の結果、送信データが受信可能でないときには、受信側のメッセージパッシング型ライブラリはバッファドレスを返さないで、エラーを送信側の計算機に通知する。

【００８４】最後に、送信側のメッセージパッシング型ライブラリは、送信データと受信されたバッファアドレスの送信を送信側の高速転送ライブラリ１３６に要求する第３のＰＵＴコマンドを発行する。高速転送ライブラリ１３６は、このデータとバッファアドレスを受信側の計算機１０２に送信する。受信側の計算機１０２内の内部高速通信専用ハードウェア１１９は、このバッファアドレスとデータを受信し、そのアドレスを有するバッファに受信したデータを直接書き込む。こうして、データの転送が終了する。

【００８５】なお、内部高速通信網１０５を介したデータ通信方法は以上の方法に限らず他の方法も使用可能である。たとえば並列計算機によっては、高速通信ライブラリはＰＵＴコマンドの他にリモートメモリ書き込みコマンドあるいはＧＥＴコマンドも実行可能である。このような並列計算機の場合には、送信側のメッセージパッシング型ライブラリ１４１が上記第３のＰＵＴコマンドを発行するのに代えて、受信側のメッセージパッシング型ライブラリ１４０がＧＥＴコマンドを発行し、受信側の高速通信ライブラリ１３５が送信側のメモリ１２５から送信データを読み出す処理を実行する。

【００８６】（８）メッセージパッシング型の通信におけるストリーム通信

図４に示されたデータ受信処理７０７，送信処理７０８の詳細を図７、８に示した具体例を適宜参照して説明する。図４において、すでに述べたように送信側のアプリケーションプログラム１０８が、関数ＥＭＵ＿ｗｒｉｔｅを呼び出した結果（７０２）、送信処理７０８が起動されると、メッセージパッシング型ライブラリ１４１に含まれた関数ＭＰＩ＿Ｓｅｎｄが呼び出される（７８１）。図７に示した例では、関数ＥＭＵ＿ｗｒｉｔｅ呼び出し時（７０２）の引数ではバッファア８０３のアドレスＳｂｕｆｆと、５０キロバイト（ＫＢ）のバッファサイズを指定すると仮定する。関数ＭＰＩ＿Ｓｅｎｄ呼び出し時（７８１）の引数が指定するバッファアドレスとバッファサイズはこれらの値に等しくされる。以下では使用する関数の他の引数の説明は簡単化のために省略する。また、指定するバッファサイズの単位はＫＢであると仮定し、関数の引数を図示するときには、この単位ＫＢは簡単化のために図示しない。メッセージパッシング型ライブラリ１４１はこの関数呼び出し７８１に応答して、データの転送のために、既に述べたように高速通信機構１１７にヘッダの送信を指示し、その後データの

送信を指示する。

【００８７】一方、図４において、すでに述べたように受信側のアプリケーションプログラム１０７が、関数ＥＭＵ＿ｒｅａｄを呼び出した結果（７０１）、受信処理７０７が起動されると、この受信処理７０７では図６に示す処理がなされる。なお、図７に示した例では、関数ＥＭＵ＿ｒｅａｄの呼び出し７０１の引数が指定するバッファアドレスはＲｂｕｆｆであり、バッファサイズは３０ＫＢであり、送信関数ＥＭＵ＿ｗｒｉｔｅの呼び出し７０２が指定したバッファサイズより小さいサイズを指定していると仮定する。

【００８８】この受信処理７０７では、まず、受信側のＴＣＰ／ＩＰエミュレーションライブラリ１１２内に設けられるデータ受信用の特定のバッファ（後に示すバッファ９０１）に受信済みでまだ受信側のアプリケーションプログラムに転送されていないデータが残っているかを判定する命令を発行する（処理２０２）。この命令は図７には示されていない。今仮定しているように最初に関数ＥＭＵ＿ｒｅａｄが実行されたときにはこの判定の結果は否定的となる。その後、送信データ検知命令ＭＰＩ＿ｐｒｏｂｅ７７１が発行され、受信側のアプリケーションプログラムに宛てて送信されようとするデータがあるかを判別する（処理２０６）。この命令は、具体的には、このアプリケーションプログラム１０７に宛てて送信されるべきデータに関する、既に説明したヘッダが高速通信機構１１６により受信済みであるか否かを判定する命令である。もしこのヘッダが送信側のアプリケーションプログラム１０８からまだ送信されていない場合には、受信処理７０８は、処理を終了し受信側アプリケーションプログラム１０７に戻る（処理２０７）。受信側アプリケーションプログラム１０７は受信が失敗したときの処理を実行する。たとえば、受信が成功するまで関数７０１を繰り返し呼び出す。

【００８９】ヘッダが送信側のアプリケーションプログラム１０８からすでに送信済みであると仮定すると、そのヘッダが指定する送信データが受信側のアプリケーションプログラムのバッファ８１２に入りきるか否かが判定される（処理２０８）。図７の例では送信データのサイズは、５０ＫＢであり、受信側のバッファ８１２のサイズは３０ＫＢであり、この判定の結果は否定的となる。このような場合に、受信側のＴＣＰ／ＩＰエミュレーションライブラリ１１２が、この送信データを受信側のアプリケーションプログラム１０７のバッファ８１２に受信するＭＰＩ＿Ｒｅｃｖ命令を発行すると、受信側のメッセージパッシング型ライブラリ１４０は、通常はＭＰＩ仕様によりこの命令をエラーとして処理するかもしくは送信データの内、受信側のアプリケーションプログラムのバッファ８１２に入りきらない分を捨ててしまう。

【００９０】これを防ぐために、本実施の形態では受信側アプリケーションプログラム１０７とリンクしている受信側ＴＣＰ／ＩＰエミュレーションライブラリ１１２内に、特別なバッファ９０１を用意し、受信側ＴＣＰ／ＩＰエミュレーションライブラリ１１２はここに送信データをこのバッファ９０１に一旦受信することを要求する。すなわち、このバッファ９０１のアドレスＥｂｕｆｆと全送信データのサイズ５０ＫＢとを指定する、関数ＭＰＩ＿Ｒｅｃｖを呼び出す（７７２）。

【００９１】この関数呼び出し７７２に応答して、受信側の高速通信機構１１６と送信側の高速通信機構１１７は、すでに述べたようにしてデータを送受信し、受信側の高速通信機構１１６は、このデータを上記バッファ９０１に書き込む。受信側のＴＣＰ／ＩＰエミュレーションライブラリ１１２はその後、この受信データの内、関数ＥＭＵ＿Ｒｅａｄ呼び出し時の引数で指定されたサイズ３０ＫＢのデータを受信側のアプリケーションプログラム１０７のバッファ８１２にコピーするメモリコピー命令（ＭＥＭＣＰＹ（Ｒｂｕｆｆ、Ｅｂｕｆｆ、３０ＫＢ））７７３を発行する（処理２０９）。その後処理はアプリケーションプログラムに戻る（処理２１１）。こうして、ＴＣＰ／ＩＰエミュレーションライブラリ１１２内のバッファ９０１に受信されたデータの一部９０６が残っている状態で、アプリケーションプログラム間の送受信が完了する。

【００９２】その後さらに送信側アプリケーションプログラム１０８がアドレスＳｂｕｆｆ’のバッファ８０４のデータ８０ＫＢを送信するために関数ＥＭＵ＿ｗｒｉｔｅ（Ｓｂｕｆｆ’，８０ＫＢ）（７９０）（図７（Ｂ））を呼び出すと、同様にして送信処理７０８が実行され、この処理の中で関数ＭＰＩ＿Ｓｅｎｄ（Ｓｂｕｆｆ’，８０）（７９１）が呼び出される。一方、受信側アプリケーションプログラム１０７もアドレスＲｂｕｆｆ’のバッファ８１３に１００ＫＢのデータを受信するために関数ＥＭＵ＿ｒｅａｄ（Ｒｂｕｆｆ’，１００ＫＢ）（７７５）を呼び出すと、これに対しても同様に受信７０７が実行される。

【００９３】この受信処理７０７は、最初の判定処理２０２における判定では、受信側のＴＣＰ／ＩＰエミュレーションライブラリ１１２内の受信用のバッファ９０１内に未転送のデータ９０６が残っていると判断される。その結果、メモリコピー命令ｍｅｍｃｐｙ（Ｒｂｕｆｆ’，Ｅｂｕｆｆ＋３０ＫＢ，２０ＫＢ）（７７６）を発行して、このデータ９０６を受信側のアプリケーションプログラム１０７内のバッファ８１３の先頭領域９０７にコピーする（処理２０３）。このメモリコピーするデータの長さは、バッファ９０１に保持されているデータの長さの内、関数ＥＭＵ＿ｒｅａｄに対する上記第２の関数呼び出しが指定するバッファ長を超えない範囲に設定される。今の場合にはバッファ９０１に保持されているデータの長さが２０ＫＢであり、受信を要求された

データの長さが１００ＫＢより小さいので、このデータ２０ＫＢが全てコピーされる。

【００９４】その後受信側アプリケーションプログラム１０７が要求する８０ＫＢの残りのデータをさらに受信するために、処理２０６が実行される。この処理２０６ではすでに説明したように、送信データがあるか否かをか調べる命令ＭＰＩ＿Ｐｒｏｂｅ（）が発行される。具体的には、送信データに対するヘッダが受信済みであるか否かが判定される。今の場合に、送信側の関数呼び出し７９１がすでに実行済みであると仮定すると、８０ＫＢの送信データがあることが判明する。その場合には、この送信データのサイズが受信側アプリケーションプログラム１０２が指定するバッファ８１３に入りきるか否かが判定される（処理２０８）。今の場合には受信側のアプリケーションプログラムのバッファ８１３の残りの領域９０８のサイズは８０ＫＢなので、送信データはバッファ８１３のこの領域９０８に入りきる。

【００９５】この結果、関数ＭＰＩ＿Ｒｅｃｖ（Ｒｂｕｆｆ’＋２０ＫＢ，８０ＫＢ）（７７８）が呼び出される。この関数呼び出しは、バッファ８１３の残りの領域９０８のアドレスと送信データのサイズ８０ＫＢを指定する。こうして、この送信データがバッファ８１３の領域９０８に直接受信される（処理２１０）。その後、受信を要求されたデータが全て受信されたか否かが判定される（処理２１２）。今の場合は判定の結果が肯定的であるので、受信処理７０７は終了し、処理はアプリケーションプログラムに戻る（処理２１１）。

【００９６】以上の手順により、複数の関数ＥＭＵ＿ｗｒｉｔｅの呼び出し（７０２，７９０）が指定する送信データをひと繋がりのデータストリームとして複数の関数の呼び出しＥＭＵ＿ｒｅａｄ（７０１、７７５）により受信する、ストリーム通信を実現することができる。

【００９７】送信側のアプリケーションプログラムが指定する送信データのサイズが受信側のアプリケーションプログラムが指定するバッファのサイズよりも小さいときでも、以下のようにしてストリーム通信が簡単に実現される。たとえば、送信側のアプリケーションプログラムが５０ＫＢのデータの送信を繰り返し要求し、受信側のアプリケーションプログラムが１００ＫＢのデータの受信を要求する場合のストリーム通信を図８を参照して説明する。

【００９８】送信側のアプリケーションプログラムが呼び出す関数ＥＭＵ＿ｗｒｉｔｅ（７０２）に対する送信処理７０８（図４）の中で、関数ＭＰＩ＿ｓｅｎｄ（７８１）が呼び出される。この関数呼び出しでは、送信側アプリケーションプログラムのバッファ８０３のアドレスＳｂｕｆｆとサイズ５０ＫＢを指定する。

【００９９】受信側のアプリケーションプログラムが呼び出す関数ＥＭＵ＿ｒｅａｄ（７０１）に対する受信処理７０７（図４）も、すでに述べたように図６に従い処理される。今の仮定では処理２０２での判定は失敗する。処理２０６において、データ検査命令７７１が実行されたときに、送信データがあると判定されたと仮定する。今の場合には受信側のバッファ８１２のサイズは、送信側のバッファのサイズより大きいので、処理２０８での判定の結果は肯定的となる。その結果、処理２１０が実行される。この処理では、送信データを受信側アプリケーションプログラムが指定したバッファ８１２に直接受信するための関数ＭＰＩ＿ｒｅｃｖ（７７４）が呼び出される。この関数呼び出しは、受信側アプリケーションプログラムのバッファ８１２の先頭アドレスＲｂｕｆｆと送信側のバッファ８０３のサイズ５０ＫＢを指定する。こうして、送信側のバッファ８０３内の全データが、受信側のバッファ８１２内の先頭の５０ＫＢの領域９０７に書き込まれる。次に処理２１２が実行される。今の場合、受信を要求されたデータのサイズは１００ＫＢであるのに対して、すでに受信されたデータのサイズは５０ＫＢである。したがって、要求されたデータの一部がまだ受信されていない。したがって、判定２１２の結果は否定的となり、残りのデータを受信するために処理２０６が再度実行される。

【０１００】もし、送信側のアプリケーションプログラムが次に関数ＥＭＵ＿ｗｒｉｔｅ７９０を呼び出せば、それに対する送信処理７０８（図４）の中で、関数ＭＰＩ＿ｓｅｎｄ（７９１）が同様に呼び出される。この関数呼び出しでも、送信側アプリケーションプログラムの次のバッファ８０４のアドレスＳｂｕｆｆ’とサイズ５０ＫＢを指定する。

【０１０１】上記処理２０６を繰り返したときに、すでに送信信側のアプリケーションプログラムが上記次の関数ＥＭＵ＿Ｓｅｎｄ７９１を呼び出していたならば、処理２０６での判定結果は肯定的となり、判定処理２０８に移る。今の場合には、受信側のバッファ８１２の残りの領域９１０のサイズは送信されようとするデータのサイズに等しいので、この判定の結果は肯定的となる。その結果、処理２１０が実行され、送信データを受信側のバッファ８１２の残りの領域９１０に直接書き込むための第２の受信関数７７９が呼び出される。この関数呼び出しでは、受信側アプリケーションプログラムのバッファ８１２の残りの領域９１０のアドレスＲｂｕｆｆ＋５０ＫＢと送信データのサイズ５０ＫＢとを指定する。こうして、処理２１２において、要求された全てのデータの受信が完了したと判断されるので、受信処理７０７は完了する。なお、上記処理２０６が繰り返し実行された時点で送信データが存在しないときには、受信処理７０７は終了し、処理は受信側アプリケーションプログラムに戻る。また、上記処理２０８が繰り返された時点で、処理２０８での判定結果が否定的であるときには、処理２０９が実行される。この処理の内容は、すでに説明したものと同じである。以上のごとく、送信側のアプリケ

ーションプログラムが発行した送信命令ＥＭＵ＿ｗｒｉｔｅが指定するバッファのサイズと受信側のアプリケーションプログラムが発行した受信命令ＥＭＵ＿ｒｅａｄが指定するバッファのサイズが異なっていても、また、送信側のアプリケーションプログラムが発行する送信命令ＥＭＵ＿ｗｒｉｔｅの数と受信側のアプリケーションプログラムが発行する受信命令ＥＭＵ＿ｒｅａｄの数が異なっていてもバッファストリーム通信が実現されることが分かる。

【０１０２】以上の説明から分かるように、本実施の形態によれば、並列計算機内部の様に、メッセージパッシング型通信方式の高速通信機構が提供されている計算機で動くアプリケーション同士が、ＴＣＰ／ＩＰを用いてデータ通信を行なう際に、高速通信機構の特徴を活かした高速通信が可能となる。また、それ以外の計算機上で動くアプリケーションとは、従来通りのＴＣＰ／ＩＰによる通信を保証する。利用者は、既存のＴＣＰ／ＩＰアプリケーションを一切変更する必要がない。

【０１０３】＜発明の実施の形態１の変形例＞本発明は、実施の形態１の内容に限定されるのではなく、以下に例示する変形例および他の変形例を含めいろいろの実施形態により実施できる。

【０１０４】（１）広域ネットワークの通信規約としてＴＣＰ／ＩＰを使用したが、これに代えて他の通信規約を用いることもできる。そのときには、ＴＣＰ／ＩＰ処理ルーチン、エミュレーションライブラリを変更する必要があるのは言うまでもない。

【０１０５】（２）実施の形態１では、メッセージパッシング型ライブラリを使用したが、これを使用しないことも可能である。このときには、エミュレーションライブラリは、直接高速通信ライブラリを呼び出すことになる。

【０１０６】（３）さらに、この通信ライブラリをなくすことも可能である。たとえば、これに代えて、専用の回路を使用することもできる。

【０１０７】（４）実施の形態１では、内部計算機は全て広域通信網に接続されると想定した。しかし、一部の内部計算機が広域通信網に接続されている場合にも同様に本発明を適用できる。

【０１０８】（５）実施の形態１では、広域通信網と内部高速通信網の両方を利用することを前提とした。しかし、本発明によるストリーム通信それ自体は、メッセージパッシング型の通信を実行可能な計算機間に適用できるものであり、したがって、このストリーム通信を実行するには、ＴＣＰ／ＩＰ通信を使用しないでメッセージパッシング型の通信のみを使用するアプリケーションプログラム間の通信にも適用できる。この場合には複数種類の通信網を使用しなくてもよい。その際には、送信側のエミュレーションライブラリは実質的には使用しない変形例も可能である。

【０１０９】＜発明の実施の形態２＞ＴＣＰ／ＩＰ処理ルーチンには従来からアプリケーションプログラムが使用可能な関数としてｓｅｌｅｃｔ関数が設けられている。そもそもソケット記述子はファイル記述子の一種として定義されている。このファイル記述子が指定するオブジェクトから、データを取得することが可能であるかどうかを調べるためのシステムコールとして、ｓｅｌｅｃｔ関数が用意されている。例えば、あるソケットから受け取り可能なデータが送信側から送られてきている（あるいは送られようとしている）かどうかも、ｓｅｌｅｃｔ関数によって調べることができる。具体的にはあるソケットを割り当てられているアプリケーションプログラムが、送信システムコールｗｒｉｔｅを発行したか否かが判定できる。ｓｅｌｅｃｔ関数では、見張ろうとするファイル記述子をビット列で指定する。このビット列の各ビットはそれぞれ個別のファイル記述子に対応しており、ビットを１にすることでファイル記述子を指定する。複数のビットを１にすれば、一回のｓｅｌｅｃｔ関数で複数のファイル記述子を同時に見張ることができる。ｓｅｌｅｃｔ関数は、見張っているファイル記述子の何れかがデータ受け取り可能な状態になるまでブロックする。

【０１１０】発明の実施の形態１で用いるＴＣＰ／ＩＰエミュレーションライブラリでは、並列計算機内部の通信時には従来のソケットライブラリを用いないため、ｓｅｌｅｃｔシステムコールでは、内部通信用のソケットからデータ受信可能であるかどうかを調べることができない。そこで、本実施の形態では、実施の形態１のごとくＴＣＰ／ＩＰエミュレーションライブラリを使用する計算機システムにおいても、アプリケーションプログラムが従来と同様にセレクト関数を利用可能にする。

【０１１１】本実施の形態では、内部ソケットテーブル４１０、４１１等に登録されているソケット記述子に対しては、内部通信専用のｓｅｌｅｃｔに相当する処理を行い、それ以外のファイル記述子に対しては、従来のｓｅｌｅｃｔシステムコールをそのまま用いる、という切り分けを行うセレクト関数ＥＭＵ＿ｓｅｌｅｃｔをＴＣＰ／ＩＰエミュレーションライブラリ１１３内に設ける。

【０１１２】ただし、内部通信用のｓｅｌｅｃｔとｓｅｌｅｃｔシステムコールは同時に並行して実行しなくてはならない。２つのｓｅｌｅｃｔを逐次に実行するのでは、例えば内部通信用のｓｅｌｅｃｔがデータを待っている間は、外部との通信用のソケットや標準入出力などがデータを受け取り可能な状態になった場合でも、それを検知することができないからである。

【０１１３】ｓｅｌｅｃｔの同時実行を疑似的に実現する手段として、内部用ｓｅｌｅｃｔとｓｅｌｅｃｔシステムコールをノンブロッキングで続けて発行することをスピンループで繰り返す、という方法が考えられる。し

かしこの方法を用いると、データが受け取り可能になるまで計算機を占有してしまい、同一計算機上で走っている他のプロセスに処理が渡らなくなってしまう。

【０１１４】これに対して本実施の形態では、１ループ毎に処理を他のプロセスに譲渡する命令を挿入するという方法を採る。この方法により、スピンループによる計算機の占有を避けることができる。

【０１１５】図９は、図３で示した接続状態において、アプリケーションプログラム１０８および１０９が送信命令１００２，１００３を発行して、アプリケーションプログラム１０７にデータを送信し、アプリケーションプログラム１０７側でそれらのデータの到着を、ｓｅｌｅｃｔ命令１００１によって見張っている様子を表す。ｓｅｌｅｃｔ命令１００１で指定しているビット列はソケットＳＡ０およびｓａ１に対応しているとする（１００４，１００５）。このうちｓａ１は内部ソケットテーブルに登録されているので、内部用ｓｅｌｅｃｔで見張る（１００８）。一方、ＳＡ０は内部ソケットテーブルに登録されていないので、ｓｅｌｅｃｔシステムコールで見張る（１００９）。ＴＣＰ／ＩＰエミュレーションライブラリで発行するｓｅｌｅｃｔシステムコールではｓａ１を見張る必要が無いので、アプリケーションプログラム１０７が発行したｓｅｌｅｃｔ命令１００１で指定されていたビット列に対し、ｓａ１に対応するビットを０にしたビット列を指定する。処理１００８と１００９はノンブロッキングに発行し、交互に繰り返す（１０１０，１０１１）。ただし、繰り返しの途中で処理を一旦、他のプロセスに譲渡する。

【０１１６】図９に示したｓｅｌｅｃｔ命令、内部用のｓｅｌｅｃｔ関数をアプリケーションプログラムが使用可能にするためには、ＴＣＰ／ＩＰエミュレーションライブラリ１１２，１１３等にはエミュレーションセレクト関数ＥＭＵ＿ｓｅｌｅｃｔが設けられ、アプリケーションプログラムは、これを呼び出して使用する。アプリケーションプログラムがＥＭＵ＿ｓｅｌｅｃｔ関数を呼び出す際には従来と同じく、それぞれ一つのソケットに対応するビットからなるビット列ａｐ＿ｂｉｔｓを指定する。ＴＣＰ／ＩＰエミュレーションライブラリは、この関数呼び出しに応答して図１０にともない処理を実行する。

【０１１７】まず、そのビット列ａｐ＿ｂｉｔｓに対し（処理１１０１）、内部ソケットテーブルに登録されているソケット記述子に対応したビットを０にするためのマスクをかける（処理１１０２）。ｉｎ＿ｍａｓｋは、内部ソケットテーブルに登録されている全てのソケット記述子に対応するビットが０、それ以外のビットが１であるようなビット列である。よって、ａｐ＿ｂｉｔｓにｉｎ＿ｍａｓｋをかけて作成したビット列ｅｘ＿ｂｉｔｓは、ａｐ＿ｂｉｔｓで指定されたファイル記述子のうち、内部通信用のソケット記述子を除いたビット列となる。その後、ｅｘ＿ｂｉｔｓを引数にしたｓｅｌｅｃｔシステムコールと、内部用ｓｅｌｅｃｔ処理をノンブロッキングで一回ずつ実行し（処理１１０３，１１０４）、もし、この処理で調べたファイル記述子の何れかがデータ受け取り可能状態であった場合にはリターンする（処理１１０６）。そうでない場合は、一旦他のプロセスに処理を譲渡し（処理１１０７）、再びｓｅｌｅｃｔ処理を繰り返す（処理１１０８）。

【０１１８】こうして、本実施の形態によれば、実施の形態１のように内部高速通信機構を併用する計算機システムにおいても、アプリケーションプログラムがｓｅｌｅｃｔ関数を利用可能になる。

【０１１９】＜発明の実施の形態３＞上記スピンループによるｓｅｌｅｃｔ関数の実現方法では、スピンループの途中に他のプロセスに処理を譲渡する処理１１０７を挿入することで計算機の占有を回避するが、同じ計算機上で処理されるプロセスの優先度が低いと、そのプロセスには処理が渡らない可能性がある。データの到着をスリープして待つブロッキングウェイトを用いればこれを避けることができるが、ブロッキングウェイトの内部通信用ｓｅｌｅｃｔ処理とブロッキングウェイトのｓｅｌｅｃｔシステムコールを、１プロセス・１スレッド上で同時に実行することはできない。

【０１２０】これに対して本実施の形態では、まず２つのスレッドを生成し、一方のスレッド上では内部用ｓｅｌｅｃｔを、もう一方ではｓｅｌｅｃｔシステムコールを実行するという方法を採る。この方法によれば、内部用ｓｅｌｅｃｔとｓｅｌｅｃｔシステムコールがそれぞれのスレッド上で独立に動作することができるため、同時にブロックしてデータを見張ることができる。

【０１２１】図１１は、図９で示したのと同じ通信状態を表している。ただし、内部用ｓｅｌｅｃｔとｓｅｌｅｃｔシステムコールの同時実行の実現方法が異なる。図１１では、これら２つのｓｅｌｅｃｔ処理は、別々のスレッドの上で実行する（１２０３，１２０４）。処理１００９と同様に、スレッド１２０３上のｓｅｌｅｃｔシステムコールでは、ｓａ１に対応するビットを０にしたビット列を指定する。

【０１２２】図１２を参照するに、本実施の形態において、アプリケーションプログラムがＥＭＵ＿ｓｅｌｅｃｔ関数を発行する際に指定したビット列ａｐ＿ｂｉｔｓに対し、内部ソケットテーブルに登録されているソケット記述子に対応したビットを０にするためのマスクをかけてｅｘ＿ｂｉｔｓを作成する処理までは図１０の処理（１１０１，１１０２）と同じである（処理１３０１，１３０２）。その後、まずノンブロッキングでｓｅｌｅｃｔシステムコール（処理１３０３）と内部用ｓｅｌｅｃｔ処理（処理１３０４）を１回ずつ行う。この処理で調べたファイル記述子の何れかがデータ受け取り可能状態であった場合にはリターンする。そうでない場合に

は、スレッドを２つ生成し（処理１３０７）、それぞれの上でｓｅｌｅｃｔシステムコール（処理１３０８）と内部用ｓｅｌｅｃｔ（処理１３０９）を実行する。これらの処理は、それぞれデータ受け取り状態になるまでブロックする。両処理のうち先にブロックが解けた方は、もう一方のスレッドをキャンセルして（処理１３１０，１３１１）リターンする（処理１３１４，１３１５）。このキャンセル処理では、スレッドを強制的に終了させるのではなく、そのスレッドがもう不要であるという印を付ける。スレッドはブロックが解けた時にこの印が付けられているかどうかを調べ（処理１３１６，１３１７）、もし付けられていれば、キャンセルされていたことになるのでそのまま消滅する（処理１３１８，１３１９）。

【０１２３】上記ｓｅｌｅｃｔ処理において、スレッド分割の前に処理１３０３および処理１３０４のノンブロッキングｓｅｌｅｃｔ処理を一回ずつ行うのは、次の理由による。もし、処理１３０１が発行される以前に内部用ソケットとそれ以外のファイル記述子が共にデータ受け取り可能な状態になっている場合、ｓｅｌｅｃｔ関数はその両方を検知できなければならない。しかし、いきなりスレッドを分割して、内部ソケットとそれ以外のファイル記述子を別々に見張り始めると、両スレッドのうち若干早く検知した方がもう一方のスレッドをキャンセルしてしまうため、片方のスレッドの状態しか検知することができない。これに対して処理１３０３および処理１３０４を実行することで、処理１３０１が発行される以前の内部ソケットとそれ以外の記述子の状態を両方とも確実に調べることができる。

【０１２４】本方式では、データの到着をスリープして待つため、実施の形態２のスピンループで待つ方法に比べると、データ検出のタイミングが遅れるが、優先度の低いプロセスに対しても、処理の妨げとなることを回避できる。

【０１２５】

【発明の効果】本発明によれば、ストリーム通信をメッセージパッシング型の通信でもって実現できる。

【０１２６】本発明の他の態様によれば、第１の通信網とそれより高速の第２の通信網に接続された計算機上で動作するアプリケーションプログラムが、第１の通信網に接続された他の計算機上で動作する他のアプリケーションプログラムとの間で第１の通信規約に基づいて通信することができ、さらに、第２の通信網に接続された他の計算機上で動作する他のアプリケーションプログラムとの間でその第２の通信網を使用した高速の通信を行うことができる。

【０１２７】さらに具体的には、上記第１の通信規約はＴＣＰ／ＩＰ通信規約を使用できる。また、上記第２の通信網を使用した通信を、メッセージパッシング型の通信とすることができる。

【図面の簡単な説明】

【図１】本発明の実施例の全体構成図。

【図２】ソケット接続のフローチャート。

【図３】ソケット接続を説明すための図。

【図４】内部・外部通信切り分け方法のフローチャート。

【図５】内部・外部通信切り分け方法を説明するための図。

【図６】ストリーム通信のフローチャート。

【図７】送受信動作に使用される命令列の一例を示す図。

【図８】送受信動作に使用される命令列の他の例を示す図。

【図９】ｓｅｌｅｃｔ機能をスピンループで実現する方法を説明するための図。

【図１０】ｓｅｌｅｃｔ機能をスピンループで実現する方法のフローチャート。

【図１１】ｓｅｌｅｃｔ機能をスレッドを用いて実現する方法を説明するための図。

【図１２】ｓｅｌｅｃｔ機能をスレッドを用いて実現する方法のフローチャート。

【図１３】従来のＴＣＰ／ＩＰ通信規約の階層図。

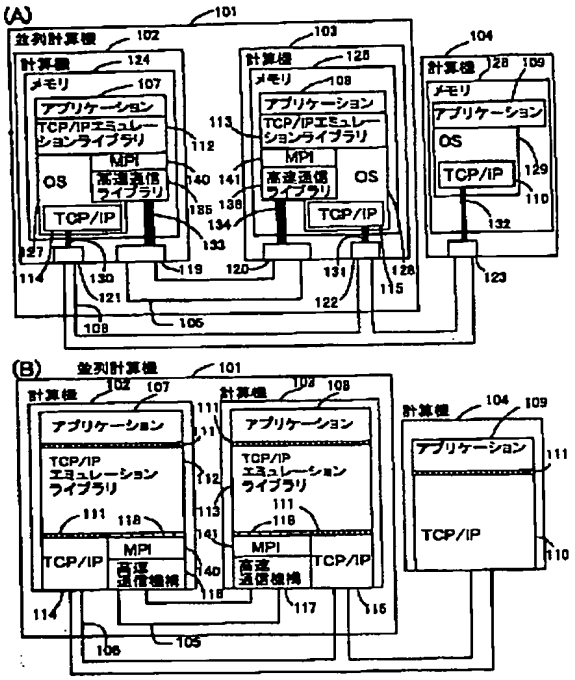
【図１４】従来のストリーム通信を説明するための図。

【符号の説明】

１０５．．．並列計算機内部通信網，１０６．．．グローバル通信網，１１１．．．ソケットアプリケーションプログラムインタフェース，１１８．．．ＭＰＩ仕様のインタフェース，１１９，１２０，１２１，１２２，１２３．．．ネットワークインタフェースハードウェア，１４０、１４１．．．メッセージパッシング型ライブラリ，４０１，４０２，４０３，４０４，４０５，４０６．．．ソケット，４０７，４０８，４０９．．．ソケットのコネクション，４１０，４１１．．．内部ソケットテーブル，６０１．．．並列計算機内部通信時におけるデータ経路，６０２．．．外部計算機との通信時におけるデータ経路，８０３，８０４，８１２，８１３．．．アプリケーションプログラムのバッファ，８０９．．．ストリーム，９０１．．．ＴＣＰ／ＩＰエミュレーションライブラリ内のバッファ。

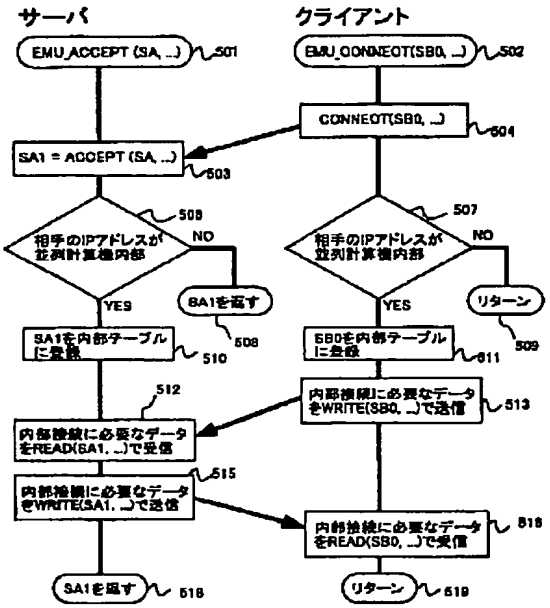
【図１】

図１



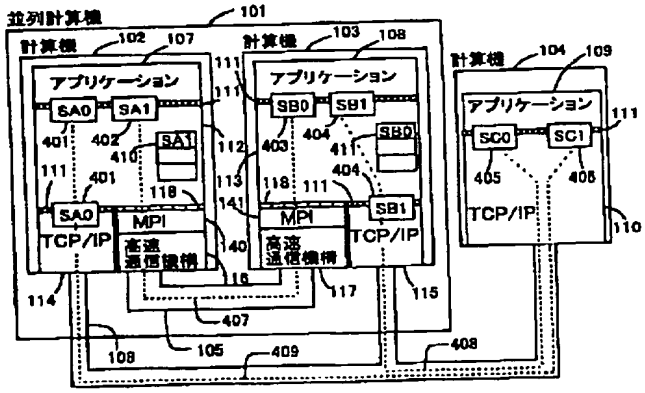
【図２】

図２



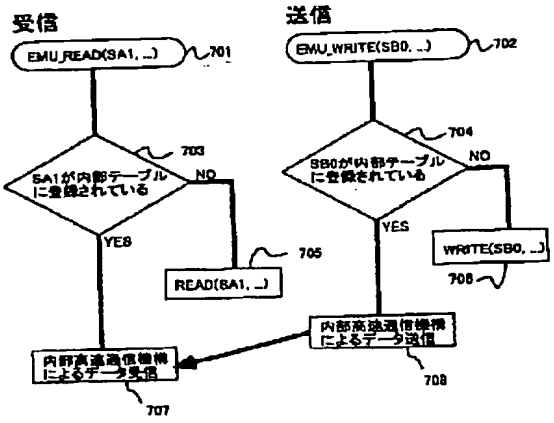
【図３】

図３



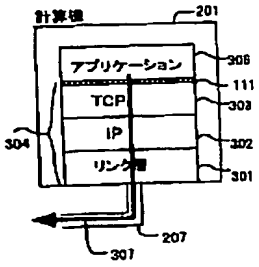
【図４】

図４



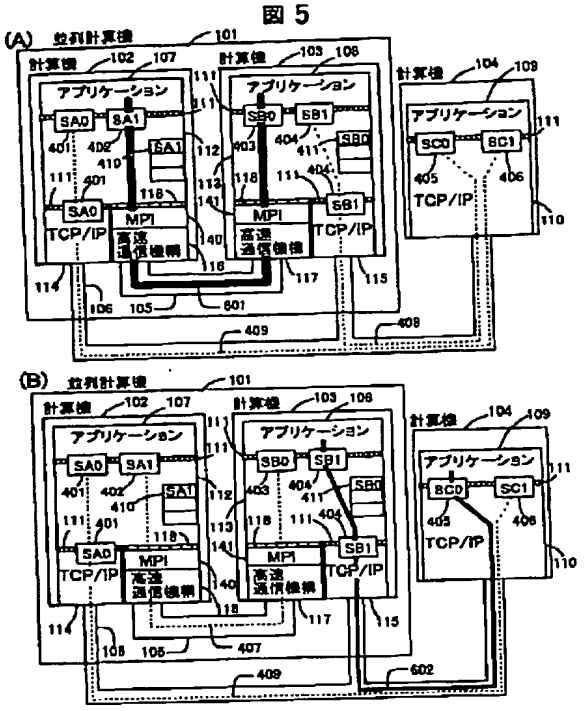
【図１３】

図１３



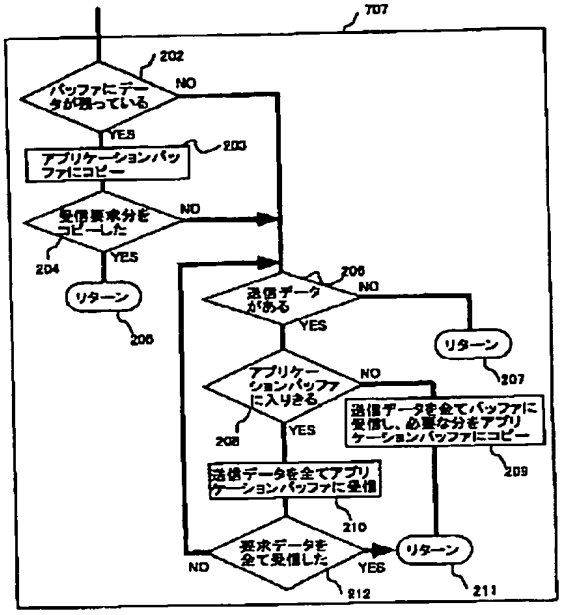
【図５】

図 5



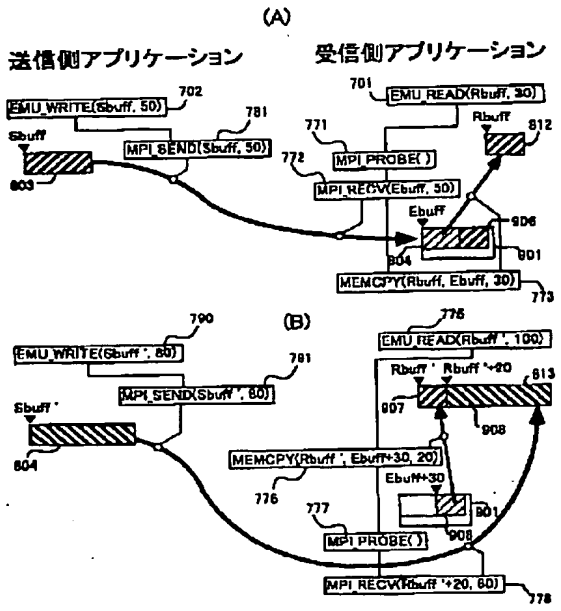
【図６】

図 6



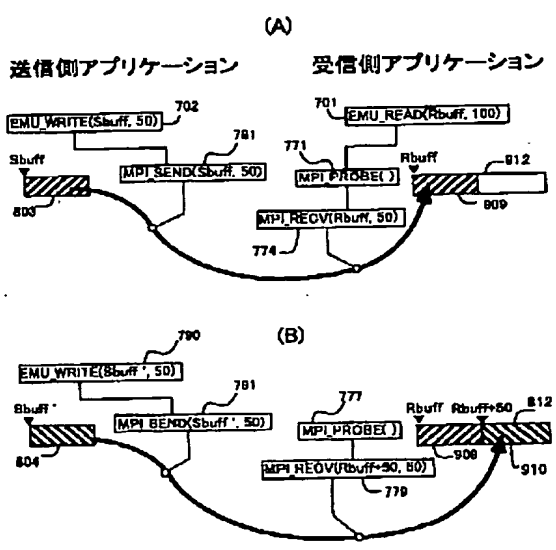
【図７】

図 7



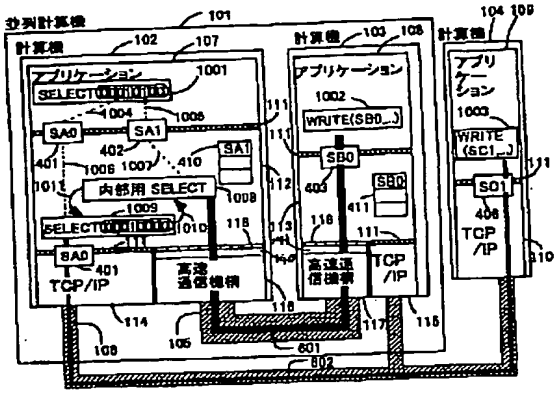
【図８】

図 8



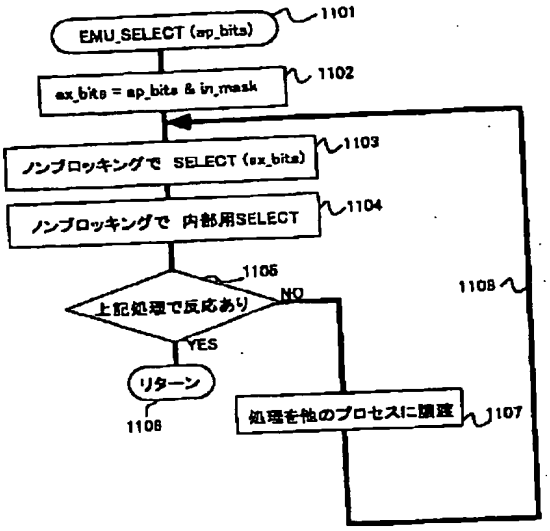
【図９】

図9



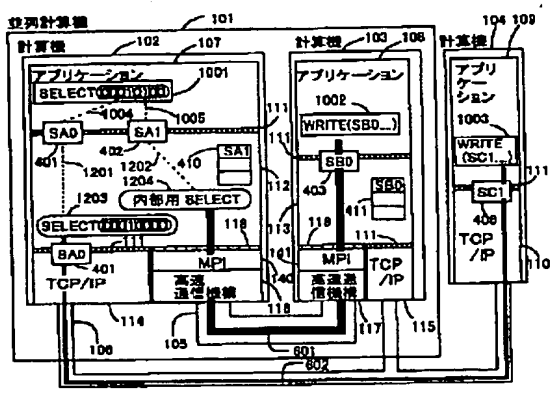
【図１０】

図 10



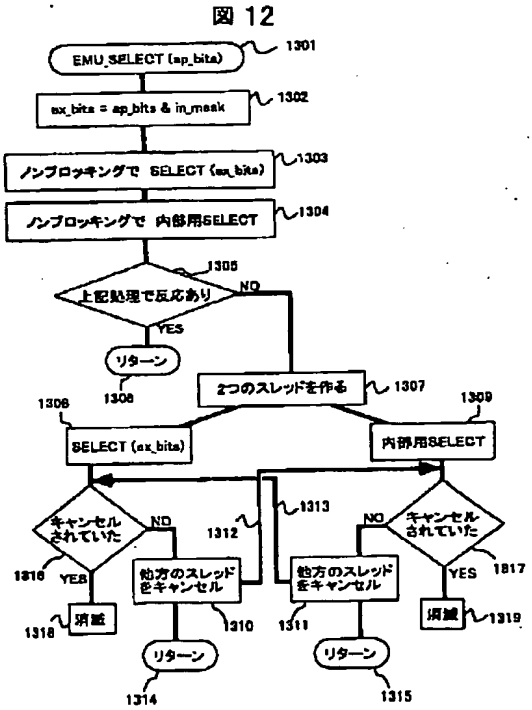
【図１１】

図 11



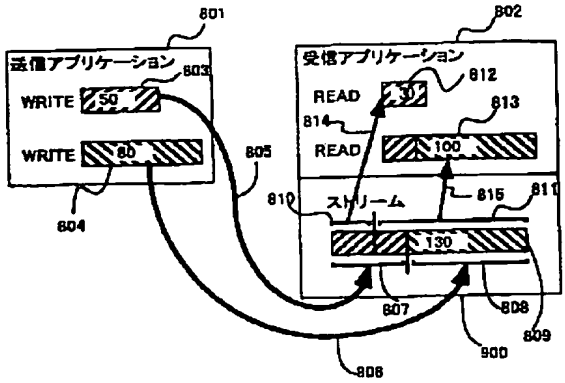
【図１２】

図 12



【図１４】

図 14


801
802
送信アプリケーション
受信アプリケーション
803
WRITE
50
WRITE
80
READ
50
812
813
814
READ
100
805
810
ストリーム
815
811
130
804
807
808
809
900
806

...S PAGE BLANK (USPTO)